NUBE DE DETECCIÓN TEMPRANA DE ACTIVIDADES TERRORISTAS

Yevheniy Kushch
Yaco Alejandro Santiago Pérez

Trabajo Fin de Grado
Grado en Ingeniería Informática
Facultad de Informática
Universidad Complutense de Madrid

Curso: 2018-2019

Director: José Luís Vázquez Poletti
Co-Directora: María Pilar Velasco Cebrián

Agradecimientos

Antes de comenzar con el documento nos gustaría agradecer a todas esas personas que han hecho posible este proyecto.

Principalmente a José Luís Vázquez Poletti, por haber sido el director de este trabajo, habernos guiado y aconsejado a lo largo de su desarrollo, siempre dándonos total libertad en la toma de decisiones, y dedicarnos su tiempo, adaptando los horarios para que pudiéramos tener reuniones de seguimiento. Ha sido un placer, y un honor haber podido trabajar junto a él.

A María Pilar Velasco Cebrián, co-directora de este TFG, agradecerle la confianza depositada en nosotros y permitirnos continuar desarrollando su idea.

A los profesores de la Facultad de Informática de la UCM, por habernos aportado tantos conocimientos a lo largo de estos años de carrera, los cuales nos han permitido llevar a cabo este proyecto.

A las Fuerzas y Cuerpos de Seguridad del Estado por su gran labor a diario en la lucha contra el terrorismo.

Y por último, agradecer a Pedro Antonio López Castelló, Comandante de la Guardia Civil de la Secretaría de Cooperación Internacional y al Cuerpo por haberse interesado en nuestro proyecto, por habernos apoyado activamente y por certificar mediante su carta de recomendación que nuestro TFG está alineado con las actividades que desarrollan.

Dedicado a todos aquellos,
que han estado a nuestro lado,
ofreciéndonos su apoyo en todo momento,
a nuestras familias, parejas y amigos

Resumen

Desde el año 2015 España se encuentra en el nivel cuatro (Riesgo Alto) de alerta antiterrorista[1]. Las Fuerzas y Cuerpos de Seguridad del Estado tienen que gestionar una enorme cantidad de datos para controlar las bandas terroristas, lo que supone un gran reto.

NDTAT es un sistema modular de recopilación y análisis de datos, cuya finalidad es ayudar a los agentes de las Fuerzas y Cuerpos de Seguridad del Estado con ese reto. Para ello, se han creado cinco módulos: el módulo de Recopilación Pasiva de datos, el módulo de Recopilación Activa de datos mediante las redes sociales, el módulo de Importación, el módulo de Visualización de datos y el Motor de Reglas.

El módulo de Recopilación Pasiva: consiste en una aplicación web, que permite a los ciudadanos colaborar, aportando información respecto a las actividades terroristas.

El módulo de Recopilación Activa: se trata de dos algoritmos de OSINT (Open Source INTelligence) que se han desarrollado para obtener los datos de Twitter y de Instagram, de forma automatizada.

El módulo de Visualización: consiste en un panel de control, donde se visualiza toda la información recopilada y analizada por el Motor de Reglas.

Palabras clave

Cloud, Sistema Modular, Sistema de Reglas, OSINT, Redes Sociales

[1] `http://www.interior.gob.es/prensa/nivel-alerta-antiterrorista`

Abstract

Since the year 2015, Spain is at the level four (High Risk) of terrorist alert. The State Security Forces, such as Police and Civil Guard, have as a task to manage a huge cuantity of data in order to control terrorist organizations, and that is a big challenge.

NDTAT is a modular system that is meant to collect and analyse data, its purpose is to help security forces with that challenge. In order to create a possible solution, we have created four modules: Passive Data Gatherer, Active Data Gatherer, Importation Module, Rules Engine and Data Visualization Module.

The Passive Gatherer consists in a web application that allows citizens to collaborate with the security forces, providing any information about any terrorist activity.

The Active Gatherer is an automatized OSINT algorithm that collects open source data from Twitter and Instagram.

The Data Visualization Module consists in a control panel that shows all of the gathered and analyzed data by the Rules Engine, as much as allows to set up all of the other modules configs.

Key words

Cloud, Modular System, Rules Engine, OSINT, Social Networks

Índice general

Índice de figuras

Introducción

Según el mapa hecho, con los datos de Wikipedia, por ESRI y PeaceTech Lab: desde el año 2016, se han producido más de cuatro mil ataques terroristas, con más de veintinueve mil víctimas mortales por todo el mundo[2]. La colaboración ciudadana junto con las nuevas tecnologías pueden ayudar a prevenir estos ataques, identificando y alertando de las posibles amenazas.

Las Fuerzas y Cuerpos de Seguridad del Estado utilizan sus bases de datos para controlar a los posibles integrantes de las bandas terroristas, pero las tecnologías utilizadas pueden ser mejoradas. Por eso, hemos decidido lanzar el proyecto "Nube de Detección Temprana de Actividades Terroristas" con la intención de crear herramientas automatizadas integrando diversas tecnologías empleadas en Data Science y OSINT.

[2] `https://storymaps.esri.com/stories/terrorist-attacks/?year=2017`

Introduction

According to the map made, with the data from Wikipedia, by ESRI and PeaceTech Lab: since the year 2016, there have been more than four thousand terrorist attacks, with more than twenty-nine thousand fatalities throughout the world. The citizen collaboration and new technologies could help to prevent those attacks, identifying and warning of possible threats.

The state security forces use their databases to control potential members of terrorist organizations, however the tecnologies they use can be improved. That is why we decided to launch this project, with the intention of creating automatized tools integrating some of the technologies used in Big Data and OSINT.

Capítulo 1. ¿Por qué NDTAT?

1.1. Motivación

El terrorismo siempre ha sido una preocupación para el Estado español. A lo largo de los últimos 70 años, se han producido numerosos atentados terroristas perpetuados por diferentes grupos armados. De distintas ideologías y procedencias, pero todos con el mismo objetivo: intimidar y sembrar el terror.

A lo largo de estos años, las Fuerzas y Cuerpos de Seguridad del Estado han investigado, combatido y mitigado estas amenazas.

Con la aparición del autoproclamado Estado Islámico, la manera de operar ha sufrido grandes cambios, como el apoyo en las nuevas tecnologías y la difusión de propaganda en redes sociales entre otros.

Del mismo modo, las fuerzas del Estado han ido modernizándose, y hoy en día cuentan con toda la información que manejan informatizada, y almacenada en grandes bases de datos. Pero explorar toda esa información, con el objetivo de cruzar datos, resultaría en una tarea tediosa y muy lenta, que consumiría mucho tiempo a los agentes.

Muchas veces, previo a cometerse un acto terrorista, se levantan distintas sospechas las cuales no son notificadas por su baja importancia, o son notificadas, pero no es posible encontrar relación entre las distintas alertas que se registran.

Es por ello, que hemos desarrollado este proyecto, con el objetivo de crear una herramienta capaz de ayudar a los cuerpos de seguridad del Estado a continuar su lucha contra el terrorismo con la ayuda de las Tecnologías de la Información, en el día a día, y en el nuevo campo de batalla que es internet.

1.2. Objetivos

El principal objetivo de este proyecto es crear una solución, que sirva a la sociedad, y aporte respuestas a problemas del mundo real, en un tema tan delicado como es el terrorismo, aplicando los conocimientos técnicos y del sector de la informática que hemos obtenido a lo largo de estos años de estudio y trabajo. Adicionalmente, desde un punto de vista más técnico tenemos los siguientes objetivos:

- Crear un sistema modular y escalable, para que en el futuro se pueda continuar desarrollando, y actualizar el sistema conforme a las necesidades futuras.
- Permitir que el sistema se pueda alimentar de información verídica, proporcionada desde otros sistemas con otros tipos de bases de datos.
- Crear algoritmos de recopilación automatizada de datos *Open Source*, para su posterior análisis.
- Desarrollar un panel de control para visualización de los datos, y con un sistema de reglas para el manejo de estos datos.
- Analizar el texto de las publicaciones obtenidas por el módulo de Recopilación Activa, mediante el análisis de sentimientos.
- Crear una red neuronal, con la idea de que vaya aprendiendo de las publicaciones, para poder detectar más fácilmente las cuentas de los implicados en el terrorismo.

- Implementar un módulo de análisis de grupos de Telegram.
- Desplegar todo el sistema en un *Cloud* fortificado y establecer conexiones seguras mediante certificados.
- Presentar el proyecto a la Guardia Civil una vez esté terminado, a la par que ofrecer nuestra colaboración, en el caso de que estuvieran interesados en cualquier aspecto.

1.3. Plan de trabajo

Para la planificación del proyecto, hemos decidido seguir una metodología clásica, en cascada, de modo que iremos desarrollando los módulos de manera secuencial. Al principio del proyecto, analizamos la extensión y la profundidad que queríamos darle, y en función de eso, previmos como distribuiríamos el tiempo para cada tarea. Las tareas troncales son la realización de los módulos principales, y la redacción de esta memoria.

En el siguiente *diagrama de Gantt* se puede encontrar la división del trabajo en tareas. Todos los tiempos que aparecen en el diagrama son estimados, por lo cual pueden sufrir cambios.

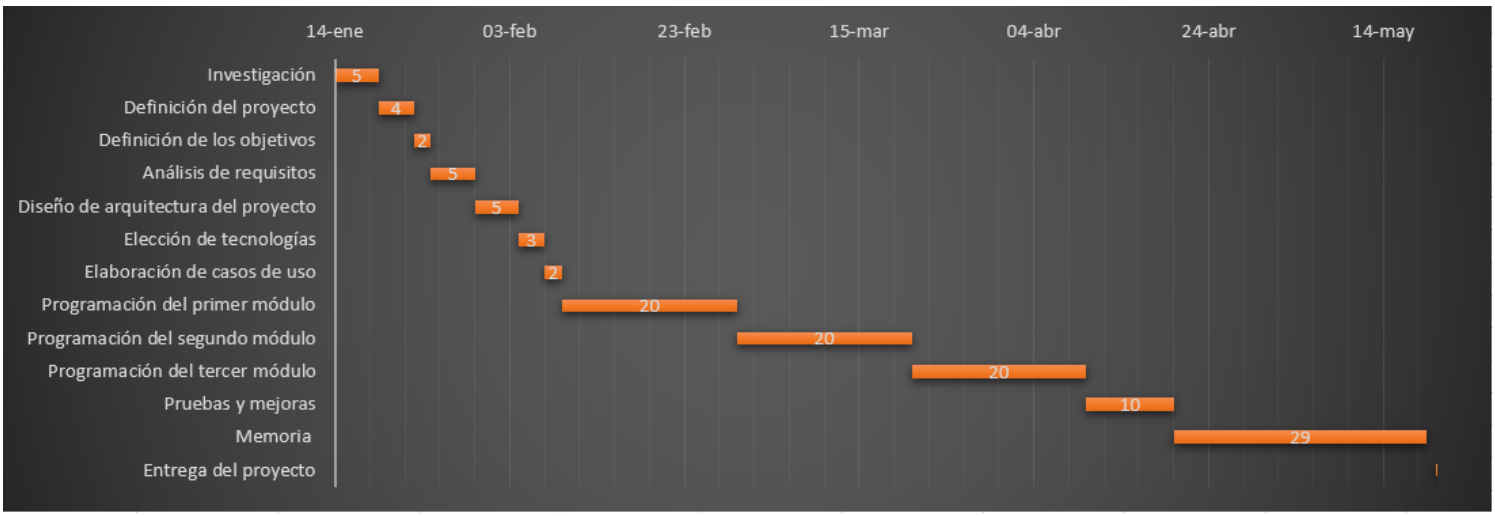


Figura 1.1: Diagrama de Gantt

Capítulo 2. Estado del arte

El País: España y otros países lanzan un fichero judicial antiterrorista europeo

España junto con Francia, Bélgica, Alemania, Holanda y Luxemburgo se han comprometido a crear un registro antiterrorista europeo[1]. Su finalidad es permitir compartir los datos de manera rápida y segura, para que en caso de un atentado, los países europeos puedan colaborar reduciendo así el tiempo de las investigaciones, e incrementando las posibilidades de llevar a los terroristas ante la justicia.

Cuadernos de la Guardia Civil (Núm. 54-2017). El uso de las nuevas tecnologías por el terrorismo yihadista

Rodrigo Lodeiro Corral, comandante de la Guardia Civil, que pertenece a la Sección de Operaciones del Estado Mayor de Mando de Operaciones de la Guardia Civil, explica en su artículo [1] que el uso de las nuevas tecnologías, con la finalidad de cometer una acción terrorista, es un problema transnacional que exige una acción coordinada e integrada entre los diferentes estados del sistema interestatal. Sin embargo, elaborar una estrategia global contra el *ciberterrorismo* tiene muchas dificultades, entre las que destaca la falta de

[1] `https://elpais.com/internacional/2018/11/05/actualidad/1541449082_914664.html`

conocimientos técnicos.

Según el comandante, las nuevas tecnologías son uno de los factores estratégicos que han sabido explotar las organizaciones terroristas con multitud de finalidades, desde el reclutamiento, la propaganda, la financiación, hasta el acopio y difusión de información con los fines terroristas. Así, las organizaciones como Al Qaeda e ISIS son las más activas en internet, teniendo multitudes de perfiles en varias redes sociales, pero sobre todo en Twitter.

Crímina (UMH): Terrorismo yihadista y nuevas tecnologías

Crímina es un centro de investigación y formación criminológica de la Universidad Miguel Hernández.

Isabel Ramos Alonso, en su artículo [2] explica que las organizaciones terroristas yihadistas han logrado profesionalizarse en el área de propaganda online y difusión de contenidos mediante Twitter, Facebook e Instagram. Por ejemplo, en junio del 2014, ISIS lanzó una campaña #AllEyesOnIsis en Twitter, mediante la cual los adeptos de la organización yihadista subían miles de fotos con el *hashtag* en diferentes puntos del mundo.

Instagram es otra de las redes sociales que más se utiliza para la propaganda. Los miembros y simpatizantes de ISIS, suben las fotos de su estilo de vida, siempre dando una imagen idealizada y atractiva, animando a los jóvenes a unirse a su causa.

Capítulo 3. Tecnologías

Después de un estudio de diferentes tecnologías, y reflexionando sobre cuáles serían las más óptimas para este proyecto, hemos optado por las que vienen a continuación.

3.1. Python

Python[1] es un lenguaje de programación interpretado, que soporta programación imperativa, POO y programación funcional. Hemos elegido este lenguaje para desarrollar nuestros scripts, por la facilidad de su uso, su licencia de código abierto y la inmensa cantidad de librerías que soporta.

Figura 3.1: Python

3.1.1. Librería xlrd

Es una librería de Python que permite a los desarrolladores extraer los datos de las hojas de cálculo de Microsoft Excel. Usamos esta librería para parsear las bases de datos en Excel y facilitar su conversión a JSON.

[1] `https://www.python.org/`

3.1.2. Módulo sqlite3

El módulo sqlite3 da la posibilidad de operar con las bases de datos SQLite mediante una DB-API 2.0. Necesitamos este módulo para *parsear* los datos guardados en SQLite y pasarlos a un JSON.

3.1.3. Módulo Twint

Twint[2] es una herramienta de OSINT (Open Source INTelligence), que permite realizar *Web Scraping* del Twitter, sin utilizar la API de Twitter, por lo cual no existe ninguna restricción a la hora de hacer peticiones. Este módulo nos da la posibilidad de personalizar las *queries* al máximo. Por ejemplo: existen las opciones para la búsqueda de *tweets* por palabras y/o *hashtags*, obtener la biografía de un usuario, obtener la lista de seguidores y a quién sigue un usuario, buscar los *tweets* escritos en un idioma específico, seleccionar el país y la geoposición, etc.

3.1.4. Módulo Pymongo

Pymongo es la librería de Python por excelencia, utilizada para interactuar con bases de datos MongoDB de manera sencilla.

3.1.5. Módulo JSON

Hemos recurrido a esta librería de Python para la importación o creación y exportación de objetos en formato JSON, estandar de la sintaxis de objetos en JavaScript.

[2] `https://github.com/twintproject/twint`

3.2. MaterializeCSS

Figura 3.2: Materialize

Materialize CSS[3] es un framework desarrollado y liberado por Google. Lo hemos elegido porque nos permite crear páginas web de manera muy rápida y con las guías de Material Design. Cabe destacar, que aunque sea un *framework* relativamente nuevo, dispone de una amplia documentación.

3.3. Google Charts

Figura 3.3: Google Developers

Google Charts[4] es una librería de JavaScript que nos permite representar ciertas informaciones y datos estadísticos de una manera más visual mediante distintos tipos de gráficas, y en contraposición a otras librerías parecidas, esta nos permite representar información de localizaciones geográficas sobre mapas visuales.

Para esta última función hemos tenido que darnos de alta en Google Developers y obtener una API KEY de geo-localización[5] para poder pintar la información sobre los mapas.

[3] https://materializecss.com/
[4] https://developers.google.com/chart/
[5] https://developers.google.com/maps/documentation/geocoding/get-api-key/

3.4. Node.js

Figura 3.4: Node.js

Node.js[6] es un framework de JavaScript de código abierto. Es multiplataforma y, principalmente, se usa para desarrollar la capa del servidor. Fue creado con el objetivo de ser útil para el desarrollo de aplicaciones de red altamente escalables.

3.4.1. Express

Express es un módulo de Node.js que nos permite el lanzamiento de un servidor web y la gestión de peticiones HTTP. Con este módulo implementamos sobre Node.js la web de Colaboración Ciudadana y el portal *Dashboard* en nuestro servidor.

3.4.2. Mongoose

Mongoose es un *Object Document Mapper* (ODM), que nos permite definir los esquemas de los objetos utilizados en la aplicación, los cuales van a estar almacenados en la base de datos MongoDB.

3.4.3. BcryptJs

Nos permite obtener el hash de un texto utilizando la librería de encriptación Bcrypt. Se ha utilizado para el cifrado de las contraseñas de los usuarios, al almacenarlas en la base de datos.

[6]https://nodejs.org/es/docs/

3.4.4. EJS

Embedded Javascript templating (EJS) es un lenguaje de marcado que utiliza tanto HTML, como Javascript, y permite crear páginas web dinámicas escribiendo scripts directamente en el cuerpo de la página.

3.4.5. Connect Flash

Es un paquete de Express que nos facilita la tarea a la hora de mostrar mensajes en pantalla en función de el resultado de las operaciones del sistema.

3.4.6. Express Session

Es un paquete de Express que se encarga de la gestión de las sesiones de los usuarios una vez han iniciado sesión correctamente. Trabaja de la mano con Cookie Parser y Passport

3.4.7. Cookie Parser

Este paquete se encarga del *desparseo* del contenido de las *cookies* que se reciben en la petición HTTP para poder interaccionar con él. Concretamente en la gestión de sesiones.

3.4.8. Passport.js

Passport es un middleware de autenticación, extremadamente flexible y modular, desarrollado especialmente para las aplicaciones construidas sobre el módulo Express. Passport nos permite gestionar tanto los inicios de sesión, como el control de acceso a diferentes partes de la aplicación web.

3.5. MongoDB

MongoDB[7] es una base de datos no relacional (NoSQL) que está basada en ficheros de tipo JSON. Es altamente escalable, flexible y admite datos heterogéneos.

Figura 3.5: MongoDB

3.6. Github

Github[8] es una plataforma de desarrollo colaborativo de *software* y *hosting* de proyectos. La plataforma está basada en Git, por lo cual es una herramienta perfecta para hacer control de versiones.

Figura 3.6: Github

3.7. LaTeX

Latex[9] es un editor destinado a la maquetación de textos, principalmente de carácter científico o técnico, que nos permite generar el documento de esta memoria con altos niveles de calidad tipográfica.

Figura 3.7: Latex

[7] https://docs.mongodb.com/
[8] https://github.com/
[9] https://www.latex-project.org/

3.8. Ubuntu

Figura 3.8: Ubuntu

Ubuntu[10] es una distribución de GNU/Linux basada en la arquitectura de *Debian*. Se trata del sistema operativo que corre el *VPS* (Virtual Private Server) sobre el cual desplegamos los distintos módulos que componen nuestro proyecto.

3.8.1. Comando CronTab

Este comando se utiliza para programar la ejecución de programas o scripts periódicamente, con las frecuencias estipuladas en *GNU/Linux*. En nuestro caso, lo hemos empleado para establecer la frecuencia de la ejecución de manera autónoma del Módulo de Recopilación Activa(4.4) y del Motor de Reglas(4.5).

3.8.2. UFW

UFW se trata de un cortafuegos que crea una capa de abstracción entre el usuario y *IPTables* de *GNU/Linux*, por lo que su configuración y la gestión de las reglas se realiza de una manera más sencilla. Nosotros lo hemos utilizado para añadir seguridad a nuestro sistema, evitando que puedan conectarse al servidor las IPs no definidas. De esta manera establecemos un perímetro adicional contra posibles intrusos.

[10]https://www.ubuntu.com/

3.9. OpenVPN

Figura 3.9: OpenVPN

OpenVPN[11] es una herramienta de código abierto que permite establecer conexiones *VPN* entre clientes y servidor de punto a punto sobre *SSL*. Utilizamos esta herramienta para añadir seguridad en el acceso a nuestro servidor.

3.10. PM2

Figura 3.10: PM2

PM2[12] es una herramienta para la puesta en producción y gestión de aplicaciones desarrolladas en NodeJs. Permite que las aplicaciones se encuentren levantadas de por vida, monitorización, generar varias instancias, y muchas acciones más. La hemos utilizado para el despliegue de nuestras aplicaciones en el servidor.

3.11. Trello

Figura 3.11: Trello

Trello[13] es una herramienta online que permite organizar las tareas en un tablero con formato *Kanban* (Pendiente, En curso, Hecho). Aunque hemos seguido una metodología clásica, nos hemos apoyado en esta herramienta, la cual nos ha permitido organizarnos y tener conocimiento de forma sencilla de las sub-tareas que hemos ido haciendo en cada una de las fases.

[11] https://openvpn.net/
[12] https://pm2.io/doc/en/runtime/overview/
[13] https://trello.com/

3.12. Tecnologías descartadas

En la planificación del proyecto, y en ciertos casos, a lo largo de su desarrollo hemos tenido que elegir entre distintas tecnologías y a continuación listaremos dichas tecnologías que han sido descartadas.

3.12.1. PHP

Para el *back-end* del módulo de Recopilación Pasiva, nos planteamos utilizar este lenguaje[14], pero fue descartado debido a la gran cantidad de vulnerabilidades, que nos haría tener que controlarlas y securizar la nube, además de que nos pareció más adecuado utilizar JavaScript con NodeJs por su alta integración con las bases de datos *noSQL*.

3.12.2. MySQL

Cuando empezamos con el proyecto, nuestra idea fue utilizar bases de datos relacionales, en concreto mySQL[15], ya que teníamos más experiencia con ellas, pero pronto nos dimos cuenta de que utilizar las bases de datos SQL no es viable para NDTAT, debido a que los datos recopilados son totalmente heterogéneos y tenemos una gran cantidad de información.

3.12.3. API de Twitter

Hemos descartado el uso de la API oficial de Twitter[16], debido a las restricciones a la hora de hacer las peticiones, y la política que tiene la empresa sobre el uso de sus servicios.

[14] https://php.net/
[15] https://www.mysql.com/
[16] https://developer.twitter.com/en/docs.html

Como alternativa, hemos encontrado la herramienta Twint (3.1.3).

3.12.4. API de Instagram

A la hora de obtener información de esta red social, nuestra primera opción fue recurrir a su API[17]. Desafortunadamente, ésta había sido extremadamente limitada en sus funciones y no nos proporcionaba la información que necesitábamos. Para solucionar este problema, recurrimos a la utilización de la técnica de *Web Scraping*.

3.12.5. Let's Encrypt

Let's encrypt[18] es una autoridad de certificación (CA) gratuito que permite implementar un certificado *SSL*, para poder hacer conexiones seguras vía *HTTPS*.

Teníamos pensado utilizarlo para generar los certificados del portal web de Colaboración Ciudadana y del *Dashboard* de representación de datos, pero no ha sido posible porque han de estar vinculados a un dominio, y no a una dirección IP.

3.12.6. amCharts

amCharts[19] es otra librería de JavaScript para la presentación de datos en gráficos y estadísticas. Pero no hay tanta documentación ni comunidad de usuarios, además de que, según qué casos y para qué funciones, es necesario comprar la licencia.

[17] https://www.instagram.com/developer/
[18] https://letsencrypt.org/
[19] https://www.amcharts.com/javascript-maps/

3.12.7. Chart.Js

Chart.js[20] es una librería de JavaScript con la que se crean gráficos con el fin de visualizar estadísticas. Descartamos esta tecnología y nos decantamos por Google Charts(3.3) debido a que la documentación nos pareció más sencilla y completa, y además, Chart.js no nos aportaba gráficos de mapas y tendríamos que recurrir a otra librería adicional.

[20]https://www.chartjs.org/

Capítulo 4. Arquitectura

En este capítulo hablaremos sobre la arquitectura general de nuestro proyecto, cómo se estructura y cómo funcionan los módulos implementados.

4.1. Arquitectura global del proyecto

Como se ve reflejado en la figura 4.1, NDTAT es un sistema basado en módulos independientes, y cada uno con una tarea determinada. Estos módulos son: el módulo de Recopilación Pasiva, que recoge los datos proporcionados por los usuarios, el módulo de Recopilación Activa de datos, que extrae la información desde las redes sociales, el Motor de Reglas, el *Dashboard* o el módulo de Representación de datos, y adicionalmente, el módulo de Importación de datos. En los siguientes puntos vamos a especificar cada uno de ellos.

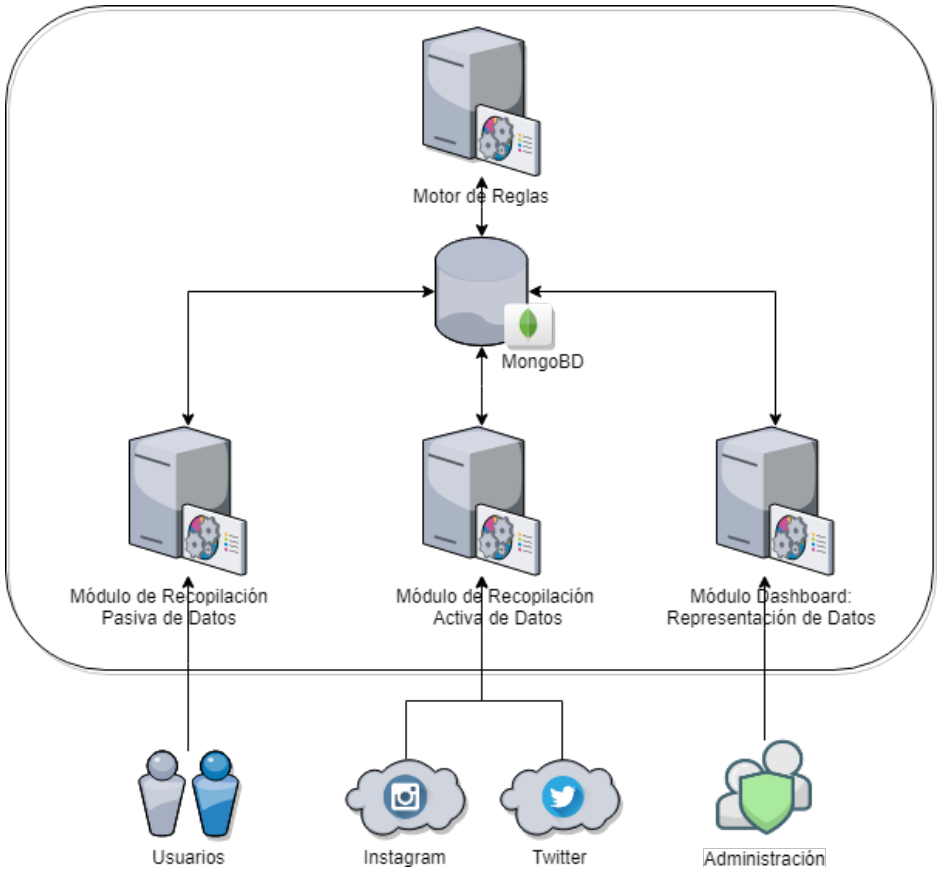


Figura 4.1: Arquitectura Global

4.2. Arquitectura de la Base de Datos

Se trata de una base de datos MongoDB (sección 3.5) no relacional. En esta base de datos almacenamos toda la información del proyecto.

Como se puede observar en el Diagrama de Colecciones (sección 4.2), las colecciones se han organizado conceptualmente, para una mejor visión y entendimiento de las mismas.

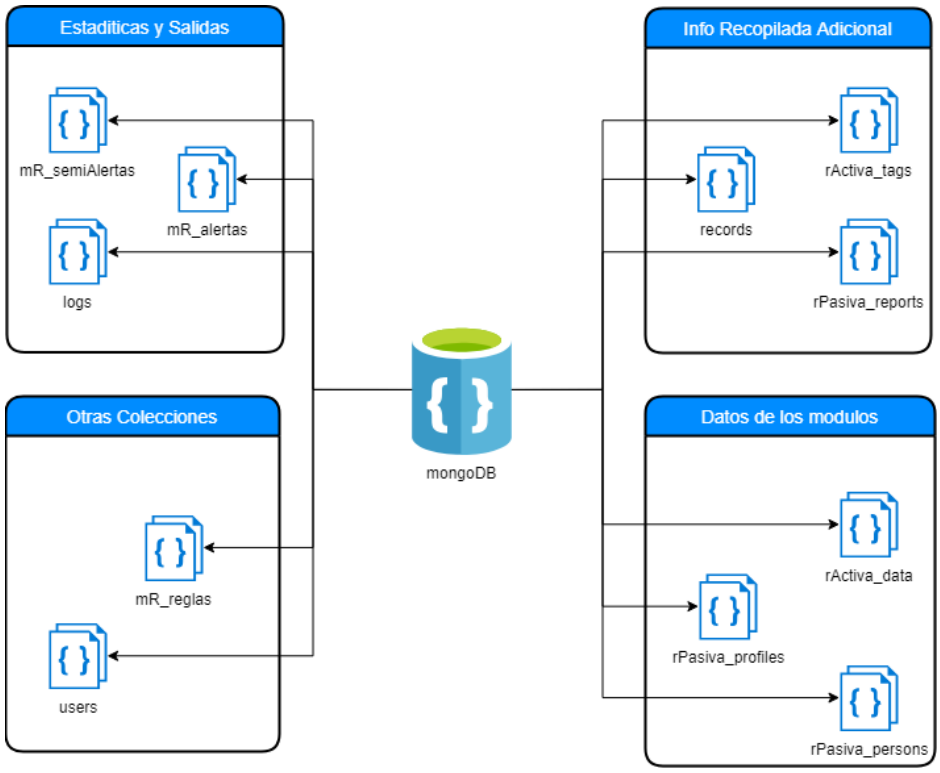


Figura 4.2: Colecciones de la BD

4.2.1. Colecciones en detalle

A continuación se explica qué información se almacena y cómo está estructurada.

Datos de los módulos

- `rActiva_data`: Almacena la información de las publicaciones recogida de las RRSS, obtenidos por el módulo de Recopilación Activa (sección 4.4).

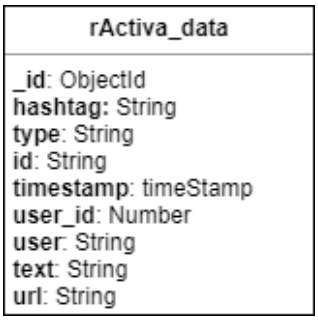
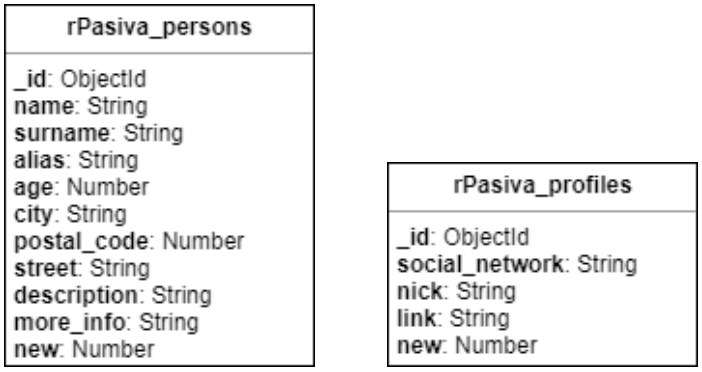


Figura 4.3: Modelo de las publicaciones recogidas en la recopilación activa

- `rPasiva_profiles`: Almacena la información de los perfiles de RRSS, obtenidos por el módulo de Recopilación Pasiva (sección 4.3).
- `rPasiva_persons`: Almacena la información de las personas obtenidos por el módulo de Recopilación Pasiva.



(a) Tipo person (b) Tipo profile

Figura 4.4: Objetos a los que hace referencia reported_id

Info Recopilada Adicional

- `rActiva_tags`: Almacena los *hashtags* que se han insertado desde la configuración del *Dashboard* (sección 4.6). Para, posteriormente, recopilar las publicaciones con dicho

hashtag en el módulo de Recopilación Activa (4.4).

Figura 4.5: Modelo de los *hashtags* de la recopilación Activa

- `records`: Almacena los datos de las personas con antecedentes penales por terrorismo. Esta información se rellena con los datos precargados, o con los obtenidos de importaciones de otras bases de datos con la ayuda del módulo de Importación (sección 4.7).

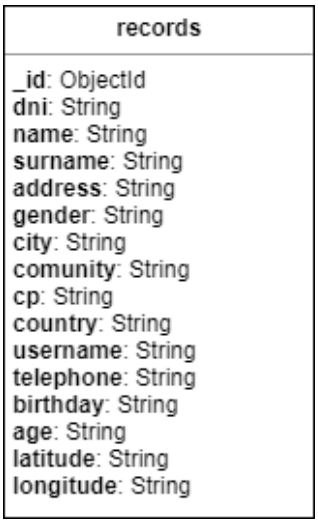


Figura 4.6: Modelo de los documentos de antecedentes

- `rPasiva_reports`: Almacena la información que se registra en el módulo de Recopilación Pasiva, cuando un usuario realiza un reporte.

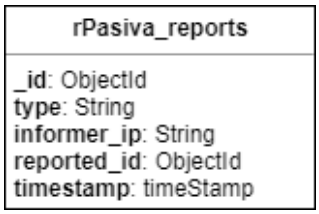


Figura 4.7: Modelo de información adicional del módulo de recogida pasiva

Estadísticas y Salidas

Esta subsección se corresponde a la información que genera el sistema. Tanto estadísticas, como las salidas del Motor de Reglas (sección 4.5).

- `mR_alertas`: Este modelo de datos es generado por el Motor de Reglas cuando se confirma que se han cumplido las coincidencias configuradas el número de veces necesario.

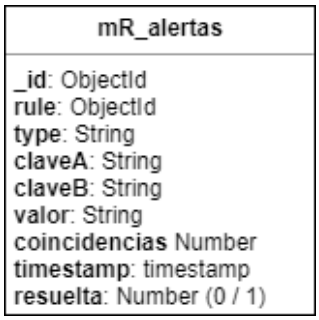


Figura 4.8: Modelo de los documentos de alerta

- `mR_semiAlertas`: Este modelo de datos también es generado por el Motor de Reglas, pero la principal diferencia respecto a las alertas convencionales es que se han producido las coincidencias pertinentes, pero no el número de veces necesario para generar una alerta. Por ello, se almacena la información hasta que se acumule el número de coincidencias mínimo para generar la alerta.

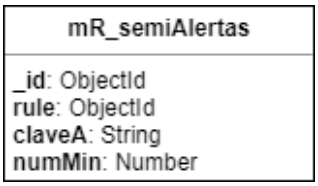


Figura 4.9: Modelo de los documentos de semi-alerta

- `logs`: En esta colección se almacenan datos numéricos de cara a la obtención de estadísticas y gráficos con una sola consulta a la base de datos.

Figura 4.10: Modelo del documento con los datos estadísticos

Otras Colecciones

- `users`: Almacena los usuarios y credenciales con sus respectivos permisos.

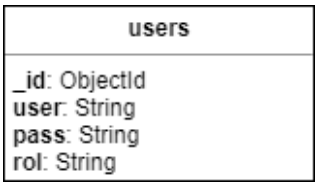


Figura 4.11: Modelo de los usuarios

- `mR_reglas`: Almacena las reglas que se definen por el administrador en el *Dashboard* (sección 4.6), que posteriormente se comprueba si se cumplen en el Motor de Reglas (sección 4.5).

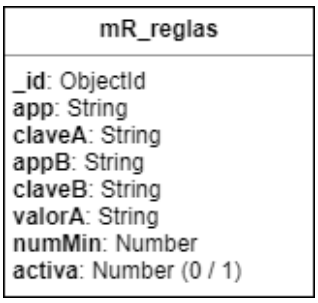


Figura 4.12: Modelo de las reglas

4.3. Módulo Recopilación Pasiva de Datos

El módulo de Recopilación Pasiva funciona como un portal de colaboración ciudadana. En la página del CNP (Cuerpo Nacional de Policia) se puede encontrar un apartado[1] para comunicar cualquier información respecto a la lucha contra el terrorismo, Guardia Civil en su página también tiene una sección[2] de colaboración.

El problema de las vías de comunicación con Guardia Civil y CNP, es que requieren el factor humano para recibir, analizar y archivar todos los correos que se les manda. Por eso, hemos pensado en crear un apartado, donde vamos a recibir la información enviada por

[1] `https://www.policia.es/terroristas/colaboracion_sp.html`
[2] `http://www.guardiacivil.es/es/colaboracion/form_contacto/index.html`

los ciudadanos. Esta información será almacenada en nuestra base de datos y será analizada por el Motor de Reglas.

En cuanto a la estructura de este módulo, hemos decidido implementarlo basándonos en el patrón de diseño Modelo-Vista-Controlador.

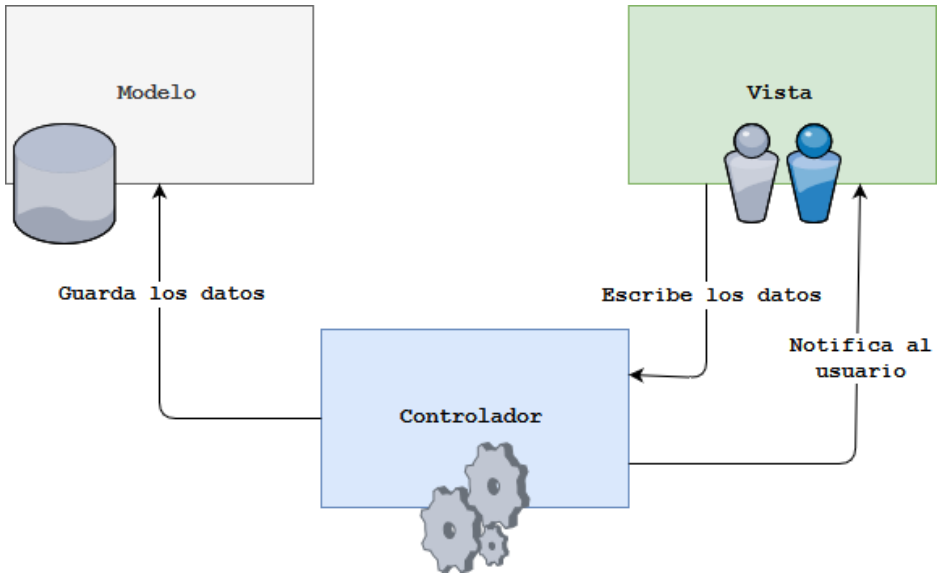


Figura 4.13: Modelo-Vista-Controlador

4.3.1. Modelo

La parte de modelo de un patrón *MVC* corresponde a la especificación y la representación de la información, y gestiona todos los accesos a dicha información.

Empezamos definiendo los datos que se van a guardar en nuestro portal. En un principio, queríamos almacenar la información únicamente de las personas, pero hemos pensado que podría ser interesante permitir aportar también la información sobre las cuentas o perfiles online, ya que en el módulo de Recopilación Activa de datos, tratamos con las cuentas de distintas redes sociales.

Para especificar los esquemas de los datos que vamos a recibir, utilizamos un mó-

dulo de Node.JS (sección 3.4), en concreto el módulo Mongoose (sección 3.4.2). En la figura 4.4, podemos observar los datos que vamos a recopilar, tanto de una persona, como de un perfil online.

Aparte, por cada reporte, se va a hacer una entrada en la base de datos, que va a contener el tipo de reporte (persona o perfil online), la dirección IP de la persona que hizo el reporte, el número identificador, y la fecha y la hora de envío (Figura 4.7).

Por último, para hacer las operaciones con la base de datos, hemos creado las funciones *SCRUD* (Search, Create, Read, Update and Delete), para cada tipo de objeto, que se va a generar a la hora de crear un reporte.

4.3.2. Vista

La vista, es una interfaz de usuario con la que interactúa el usuario. Hemos elegido el *framework* de MaterializeCSS (sección 3.2) para la creación del *front-end*. Para este módulo, hemos creado tres vistas:

- Vista principal: Aquí, es dónde se podrá elegir entre informar sobre una persona, o una cuenta de una red social.

Figura 4.14: Vista principal

- Vista persona: Esta vista contiene el formulario para informar sobre una persona. Entendemos que un ciudadano puede no tener todos los datos que le pedimos sobre el sospechoso, por lo cual, todos los campos son voluntarios. Sin embargo, para hacer la aportación, es obligatorio rellenar al menos cuatro campos, además de aceptar las condiciones.

Figura 4.15: Formulario de persona

- Vista perfil online: Esta vista sirve para informar sobre las cuentas de distintas redes sociales, que pueden estar involucradas en reclutamiento, propaganda y otras actividades relacionadas con el terrorismo. El usuario deberá rellenar todos los campos para enviar la información.

Figura 4.16: Formulario de perfil

4.3.3. Controlador

La parte de controlador, corresponde a eventos e invoca peticiones al modelo cuando un usuario hace una acción en la vista. También, modifica la vista, en nuestro caso para notificar al usuario que la operación ha tenido éxito, o que faltan datos por rellenar. A continuación, vamos a especificar las operaciones que hace nuestro controlador.

- Obtención de IP: Se utiliza la API de ipify[3] para conseguir la dirección IP del usuario que genera un informe.
- Comprobación de campos vacíos: En cada formulario, al intentar enviar la información, calcula el número de campos vacíos, y en caso de que no haya al menos cuatro campos con datos en el formulario de persona, o en caso de que haya algún campo vacío en el formulario de perfil online, salta un aviso informando que no se puede efectuar la

[3] `https://www.ipify.org/`

operación.

- Validación de código postal: Se comprueba mediante expresiones regulares que el código postal sea válido.
- Comprobación de *checkbox* de aceptación de condiciones: Comprueba si el *checkbox* está seleccionado y activa el botón de enviar.
- Creación de objeto persona: Se recogen los datos de los campos rellenados y se crea un objeto persona.
- Agregación de persona: Se manda al modelo el objeto y se notifica al usuario que la operación ha tenido éxito.
- Creación de perfil online: Se recogen los datos de los campos del formulario de perfil online y se crea un objeto perfil.
- Agregación de perfil: Se manda al modelo el objeto creado y se notifica al usuario que la operación ha tenido éxito.

4.4. Módulo Recopilación Activa de Datos

La función de este módulo es recopilar la información de fuentes abiertas, en este caso, vamos a obtener la información de Twitter y de Instagram. Hemos elegido estas redes sociales por su popularidad entre la gente joven, y por la inmensa cantidad de usuarios que las utilizan. Según el estudio[4] realizado por las empresas *We are Social* y *Hootsuite*, Instagram tiene más de 1.000 millones de usuarios activos, y el Twitter lo utilizan más de 329 millones de usuarios. Por eso, las redes sociales se han convertido en un medio perfecto

[4]https://www.slideshare.net/DataReportal/digital-2019-global-digital-overview-january-2019-v01

para hacer la propaganda y apología al terrorismo en busca de la viralización de estas ideas, y captación de nuevos seguidores.

Esto va en contra de las políticas de ambas redes sociales, pero la única medida que se toma para combatir estas publicaciones es su eliminación, y el posible bloqueo de cuenta. En ningún caso se notifica a las autoridades sobre estos comportamientos.

Por ese motivo, hemos pensado en desarrollar este módulo, cuyo objetivo es buscar y registrar este tipo de publicaciones para posteriormente analizar la información con ayuda del Motor de Reglas, generando las salidas que puedan ayudar a las FFCCSE en la lucha contra el adoctrinamiento, y el reclutamiento terrorista.

Para obtener las publicaciones de las redes sociales, al principio, hicimos dos *scripts* diferentes, uno para cada red social. Esta opción fue descartada, ya que va en contra de la idea de la máxima modularidad de nuestro proyecto. Por eso, hemos decidido unificar los dos algoritmos, incluyéndolos en un mismo ejecutable. Por lo cual, en vez de ejecutar cada script por separado, se ejecuta el script principal, que a su vez hace una llamada al recopilador de Instagram, y luego al recopilador de Twitter. De tal modo, dejamos la posibilidad de incluir algoritmos de análisis de otras redes sociales, si surge esa necesidad en el futuro.

En cuanto a funcionamiento del módulo, las búsquedas son automatizadas, se activa cada día y recoge las publicaciones de las últimas 24 horas. En la figura 4.17, podemos observar el flujo de ejecución. Primero, se conecta a la base de datos para obtener la lista de *hashtags* por los que va a realizar la búsqueda. En nuestro caso utilizamos los *hashtags* que más suelen aparecer en las publicaciones terroristas. Una vez obtenida la lista, se hace una llamada a los constructores de las clases de recopilador de Instagram y de Twitter. A continuación se recorre la lista y se ejecuta la búsqueda en las dos redes sociales. Finalmente, se insertan los resultados en la base de datos junto con los `logs` que nos permiten monitorizar las métricas de ejecución del módulo.

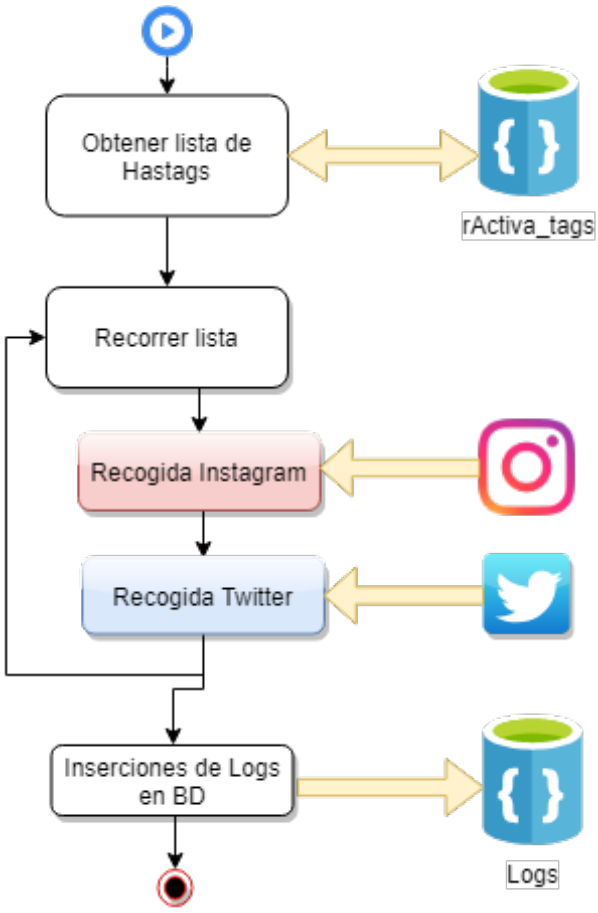


Figura 4.17: Flujo de ejecución del módulo activo

La estructura de la aplicación sería la siguiente:

```
├── model/
│   └── rActiva_data.py      # Define el modelo de los datos
├── InstagramScrapper.py     # Script de WebScraping
├── InstagramRecopiler.py    # Script de procesamiento
├── TwitterRecopiler.py      # Script de recopilación
└── App.py                   # Aplicación
```

Figura 4.18: Estructura del módulo de recopilación activa

La estructura de los datos generados será la siguiente:

- `id`: Identifica la publicación.
- `hashtag`: Palabra clave con la que se ha encontrado.
- `type`: RRSS de la que se obtiene
- `timestamp`: Fecha en la que se publicó.
- `user_id`: Identificador del usuario propietario.
- `user`: *Nick* del usuario.
- `text`: Pie de foto de la publicación, en la cual se encuentran los textos y *hashtags*.
- `url`: Dirección web para acceder a ella.

4.4.1. Recopilación de datos del Instagram

Para desarrollar esta parte del módulo, inicialmente, habíamos pensado en la utilización de la propia API[5] de la red social, pero no ha sido posible, debido a que el número y el tipo de las peticiones permitidas son extremadamente limitados desde mediados de 2018, y no son suficientes para llevar a acabo nuestro proyecto. Como una alternativa, documentándonos mediante el libro *Practical web scraping for data science: best practicesand examples with Python* [3] y el blog de *Edmund Martin* [4], decidimos utilizar las técnicas de *WebScraping*, que consisten en hacer peticiones a la página web simulando ser un navegador, y obteniendo como respuesta los archivos `html`. Las peticiones se realizan pasando como parámetro, dentro de la `URL`, los *hashtags*.

[5] `https://www.instagram.com/developer/`

Procedimiento: Por cada *hashtag* de la lista, se genera una url a la que se va a hacer una petición `http`. La respuesta viene en forma de `html`, o cuerpo de página. Ese `html` contiene todas las publicaciones que llevan incluido el *hashtag* de la petición. Cada publicación contiene una serie de datos, por lo cual se procede a recorrer cada una de ellas recolectando los datos más relevantes.

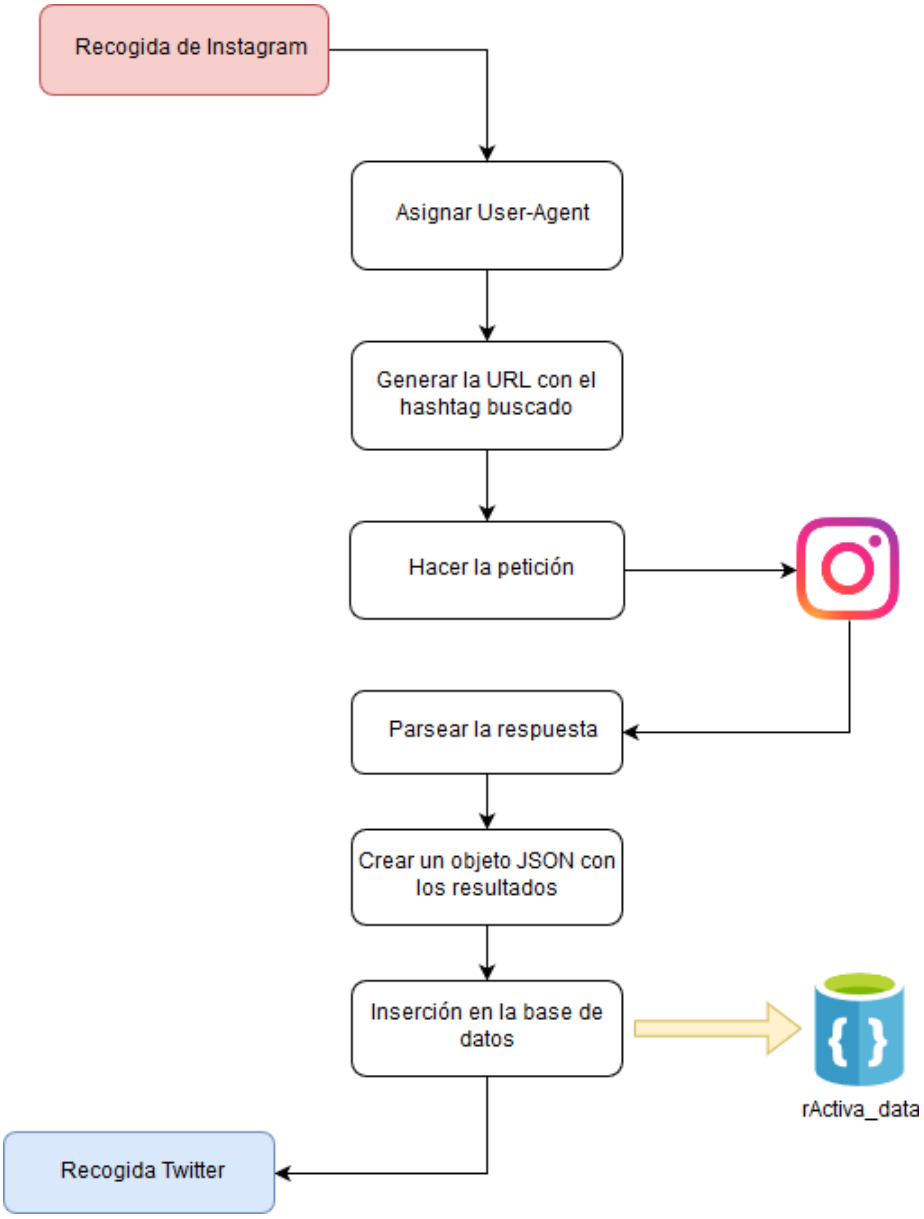


Figura 4.19: Diagrama de flujo: recogida de Instagram

4.4.2. Recopilación de datos del Twitter

Igual que en el caso de Instagram, la API de Twitter es muy limitada, por lo cual tuvimos que buscar otro método de recopilación de *tweets*. Después de analizar una serie de alternativas, encontramos el módulo `twint` (sección 3.1.3), que encaja perfectamente en esta parte del módulo.

Procedimiento: Una vez obtenida la lista de *hashtags* de la base de datos y ejecutado el algoritmo de recopilación de Instragram, se crea un objeto de configuración del módulo `twint`. Para ello, es necesario elegir y asignar valor a los parámetros indispensables para ejecutar las búsquedas. En nuestro caso necesitamos los siguientes parámetros:

- `Search`: La palabra o el *hashtag* por el cual vamos a ejecutar la búsqueda.
- `Store_csv`: Permite guardar los resultados de la búsqueda en un `csv`.
- `Output`: Indica dónde queremos guardar los resultados.
- `Since`: Desde qué fecha de publicación va a ejecutar la búsqueda.
- `Show_hashtags`: Para guardar otros *hashtags* que contiene la publicación encontrada.

Teniendo todos estos datos, se recorre la lista de *hashtags* y se hace una búsqueda por cada elemento de la lista. Seguidamente, se guardan las publicaciones en los ficheros CSV, y de estas publicaciones se crean los objetos JSON, que serán insertados en la base de datos. Se podría prescindir de los fichero CSV y tratar los datos directamente, pero las búsquedas generan una gran cantidad de resultados, por lo cual, ocuparía una parte considerable de la memoria.

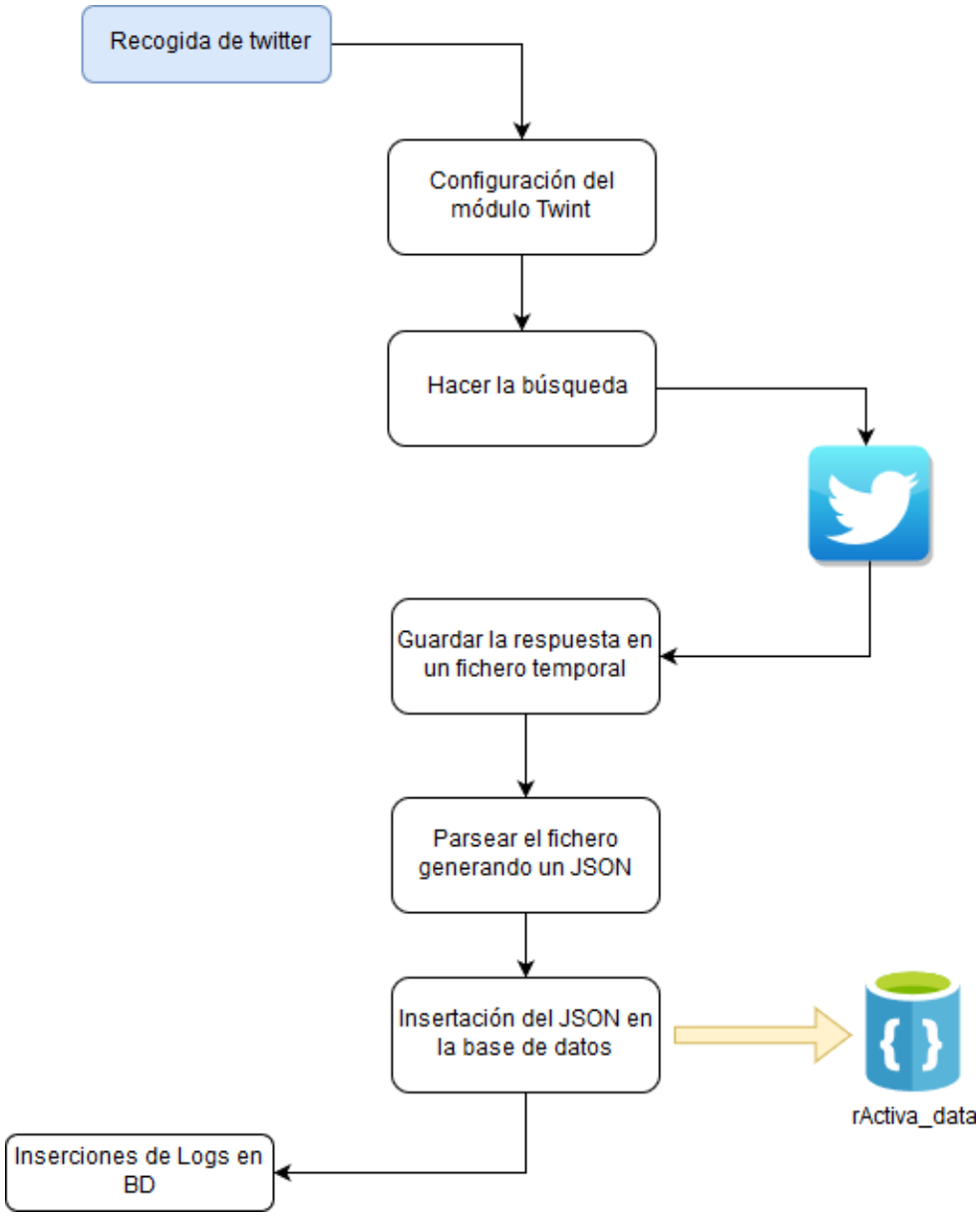


Figura 4.20: Diagrama de flujo: recogida de Twitter

4.5. Motor de Reglas

El desarrollo de este módulo se ha realizado mediante el lenguaje de programación Python (sección 3.1) y hemos empleado la librería Pymongo (sección 3.1.4) para llevar a cabo la interacción con la base de datos.
Las reglas en nuestro sistema consisten en un conjunto de requisitos, definidos por el administrador, que se tienen que cumplir un número establecido de veces para generar una alerta y notificar de estas coincidencias mediante el *Dashboard*.

4.5.1. Estructura

```
├── config.json              # Json que guarda ciertas configuraciones
├── Coincidencias.py         # Algoritmia del módulo
│
└── main.py                  # Script principal
```

Figura 4.21: Estructura del Motor de Reglas

4.5.2. Modelo de datos

En este módulo se interactúa con la gran mayoría de modelos de datos que hemos presentado a lo largo del documento, y más en detalle en la sección 4.2. Es por ello por lo que aquí únicamente nos centraremos en los tres más importantes:

- `reglas`: Modelo de la sección 4.12 que define qué campos han de coincidir entre qué módulos, en el caso de ser una regla de tipo global, o si el valor indicado se encuentra registrado en la BD de tipo única, y cuantas veces se ha de producir dicha coincidencia para generar la alerta.

- `alertas`: Modelo de la sección 4.8, de dato que se genera cuando se han sobrepasado el número necesario de coincidencias de una regla.
- `semi-alertas`: Modelo de la sección 4.9, de dato que se genera y se va actualizando cuando se encuentran coincidencias de una regla pero no ha superado el número necesario de coincidencias como para generar una alerta. Se cuenta el número de coincidencias acumuladas, y en el momento que se supere el mínimo necesario, se borra la semi-alerta y se genera una alerta.

4.5.3. Funcionamiento

- Recupera la lista de reglas a aplicar que se encuentren marcadas como activas.
- Recupera la lista de nuevos perfiles y de nuevos usuarios desde la ultima ejecución.
- Recorre todas las reglas, mira sobre qué datos actúa dicha regla.
- En función del tipo de datos, y el tipo de regla que se trate, consulta en una u otra colección, por si se ha producido alguna coincidencia, las cuales cuenta.

En la fase de generación de Alertas/Semi-Alertas:

- Comprueba si ya está registrado en semi-alertas (coincidencias que se han dado, pero aun no han sido el número suficiente para generar una alerta).
- Si lo está, suma las coincidencias actuales con las previas de la semi-alerta.
- Se comprueba si el número de las coincidencias es suficiente para generar una alerta, en caso afirmativo se crea una alerta.
- Si ya existía semi-alerta, se actualiza.

- Si no existía semi-alerta, se crea.
- Por último, en cada caso, se marca el campo `new` a 0, para que el registro ya no figure como nuevo en la base de datos, y no se vuelva a recorrer en la siguientes ejecuciones.
- Finalmente, se devuelve el número de coincidencias totales que se han encontrado durante la ejecución.

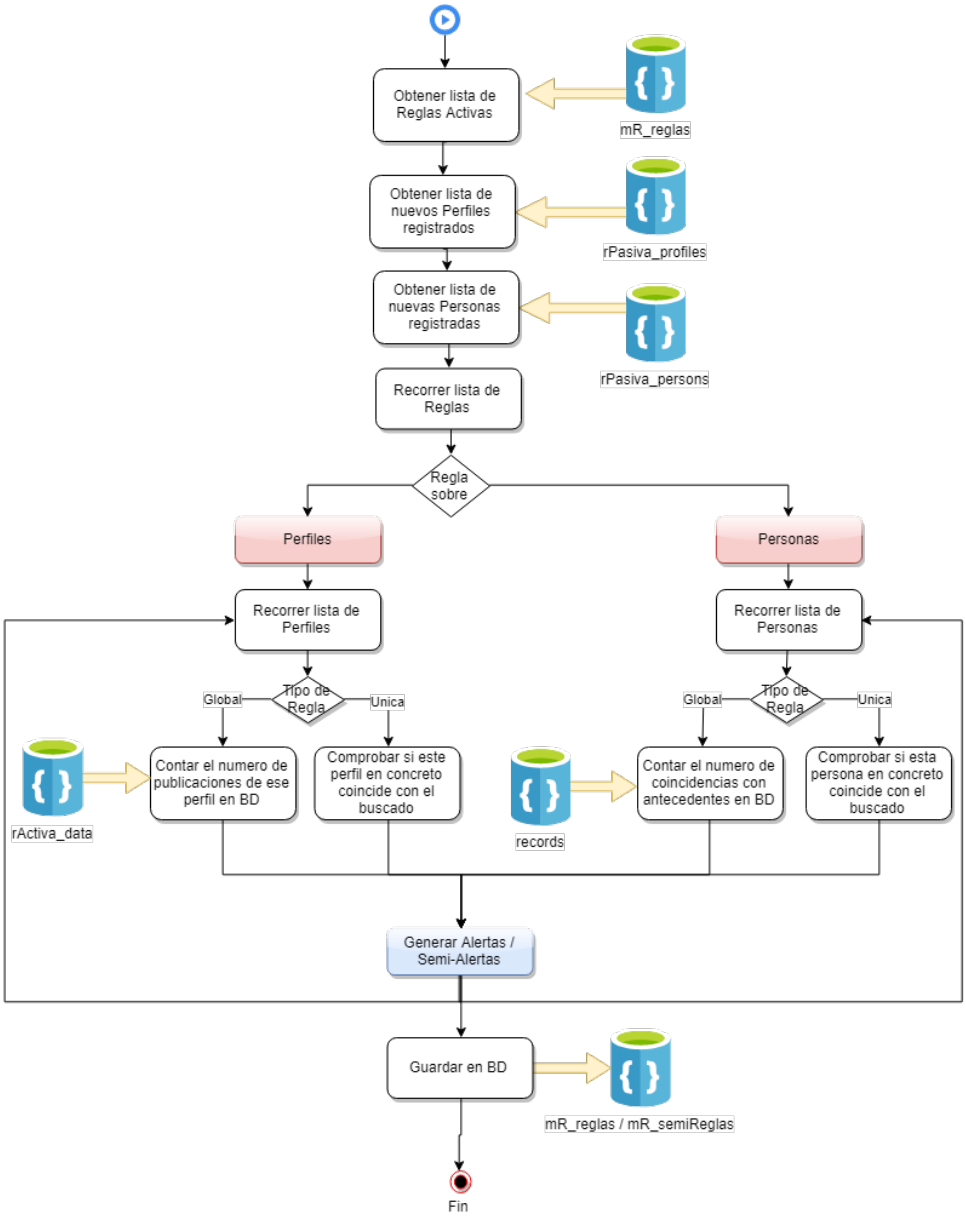


Figura 4.22: Diagrama de flujo: Motor de Reglas

4.6. Módulo Dashboard: Presentación de Datos

Este módulo ha sido programado en NodeJs. Para ello, hemos utilizado los módulos Express (sección 3.4.1) como *framework web*, Mongoose (sección 3.4.2) para facilitar la interacción con la base de datos MongoDB, BcrytJs (sección 3.4.3) para garantizar la seguridad almacenando las contraseñas de los usuarios *hasheadas*, y PassportJS (sección3.4.8) para la gestión de sesiones de los usuarios.

La estructura de archivos es la siguiente:

```
├── mongodb/
│   └── auth/                 # Contiene los archivos para la autenticación
│   └── controllers/          # Controladores de cada tipo de modelo
│   │                         #   de dato de la BD.
│   └── models/               # Modelos de datos de la BD
│   └── routes.js             # Archivo que define los handlers de
│                             #   las peticiones a BD.
│
├── public/                   # Directorio con todos los htmls
│   └── *.html                # Archivos HTML que conforman partes de las vistas
│   └── *.ejs                 # Archivos EJS que conforman las vistas
│   └── assets/               # Directorio con ficheros estáticos
│       └── css/
│       └── img/
│       └── js/
├── config.json               # Archivo donde se almacenan variables
│                             # de configuración.
├── package.json
├── package-lock.json
└── server.js                 # Servidor que lanza la aplicación
```

Figura 4.23: Estructura del módulo de Dashboard

4.6.1. Modelo

Los modelos de datos se encuentran especificados en el directorio `/mongodb/models/`, y comparten estructura con los modelos definidos para la base de datos en la sección 4.2:

- `log`: Objeto con los datos estadísticos de carácter general que se muestran en el *Dashboard* (Figura 4.10).
- `person`: Modelo de dato referente a una persona (Figura 4.4).
- `rActivaData`: Modelo de dato referente a una publicación de las RRSS (Figura 4.3).
- `rActivaTag`: Objeto que representa un *hashtag* del cual se van a buscar publicaciones en el módulo de recopilación activa (Figura 4.5).
- `report`: Objeto que representa una alerta generada por el Motor de Reglas (Figura 4.7).
- `user`: Objeto que representa a un usuario del sistema con sus respectivos datos y rol.

4.6.2. Vista

La vista está basada en un ejemplo de *Dashboard* desarollado por Mustafa Omar[5]. Para este módulo hemos desarrollado siete vistas principales.

- `Log-in`: Pantalla de inicio de sesión.

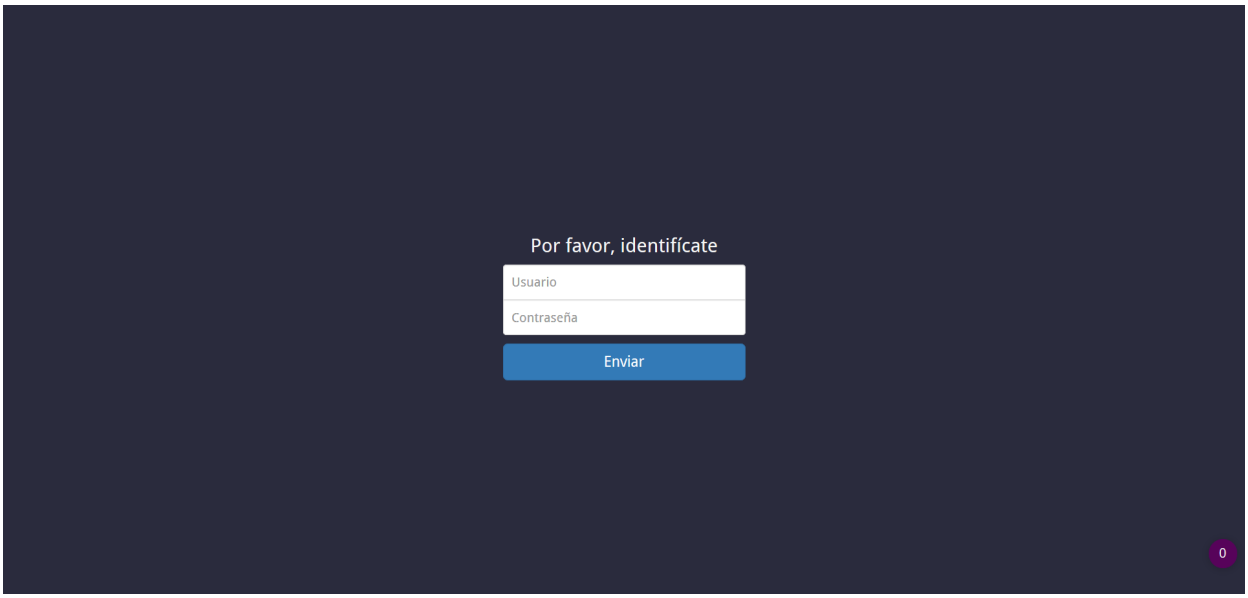


Figura 4.24: Vista log-in

- `Inicio`: Vista inicial, donde se presenta un tablero con información general:
  - Número de denuncias generadas por el módulo de recopilación pasiva de datos.
  - Número total de publicaciones recopiladas por el módulo de recopilación activa de datos.
  - Número de alertas revisadas.
  - Número de *hashtags* por los que se hace la búsqueda.
  - Número de alertas pendientes por revisar.
  - Número de reglas aplicadas.
  - Número de publicaciones recopiladas en la última ejecución del módulo de recopilación activa de datos.

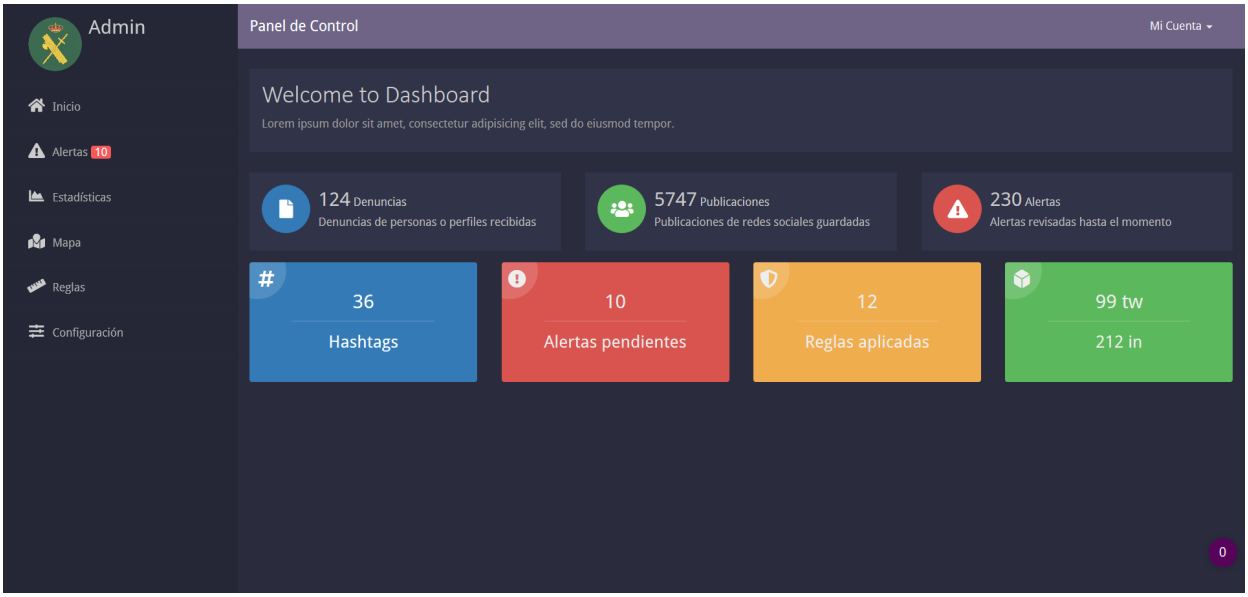


Figura 4.25: Vista inicio

- `Estadísticas`: Sección con estadísticas detalladas, presentadas en diferentes tipos de gráficos:
  - Gráfico con las cinco ciudades donde se generan más reportes.
  - Gráfico con los resultados de los últimos cinco de días de recopilación activa de datos, para ver la tendencia del número de publicaciones.
  - Gráfico con el porcentaje de cada tipo de reporte del total del módulo de Recopilación Pasiva de datos, para saber si se denuncia más a las personas o perfiles online.
  - Gráfico con el porcentaje del número de publicaciones recopiladas de cada red social, para saber en qué red social los terroristas están más activos.
  - Mapa de España con puntos térmicos, que muestran la cantidad de personas con antecedentes en cada ciudad.

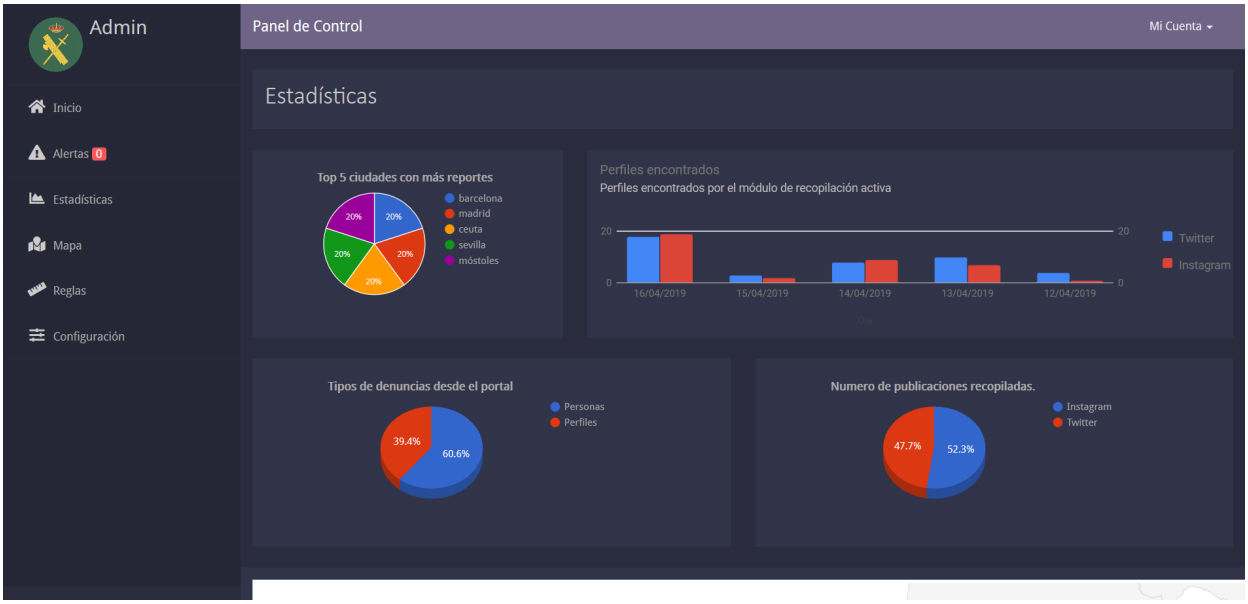


Figura 4.26: Vista estadísticas

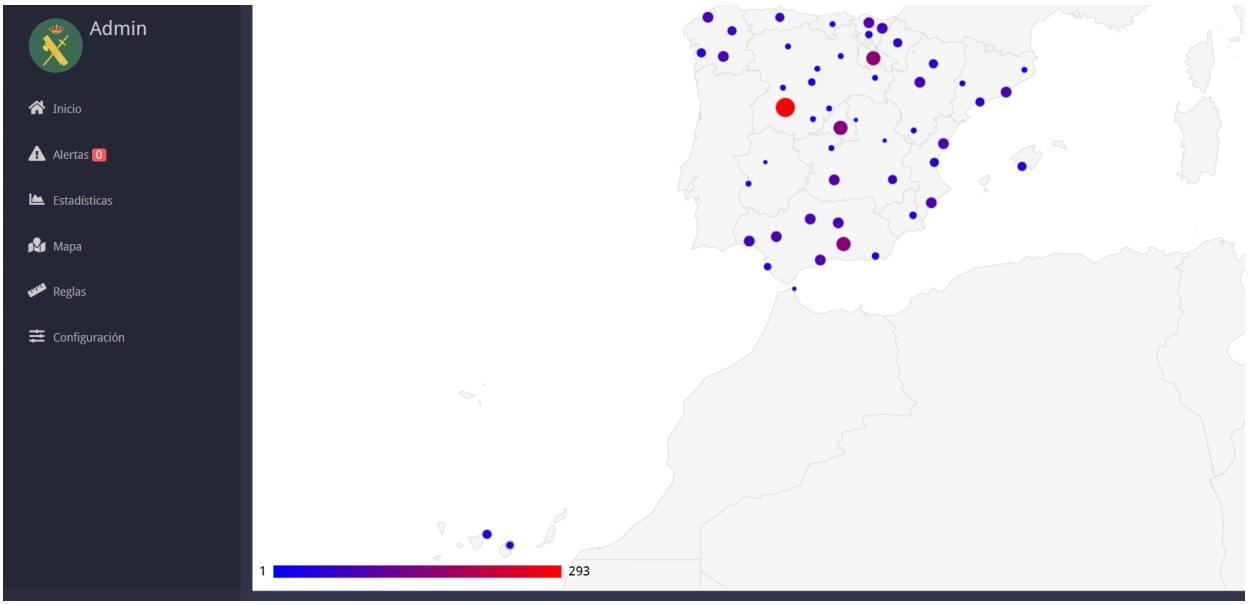


Figura 4.27: Vista estadísticas II

- `Mapa`: Sección donde se podrá encontrar un mapa de *Google Maps* con las localizaciones de domicilios de personas con antecedentes por terrorismo.

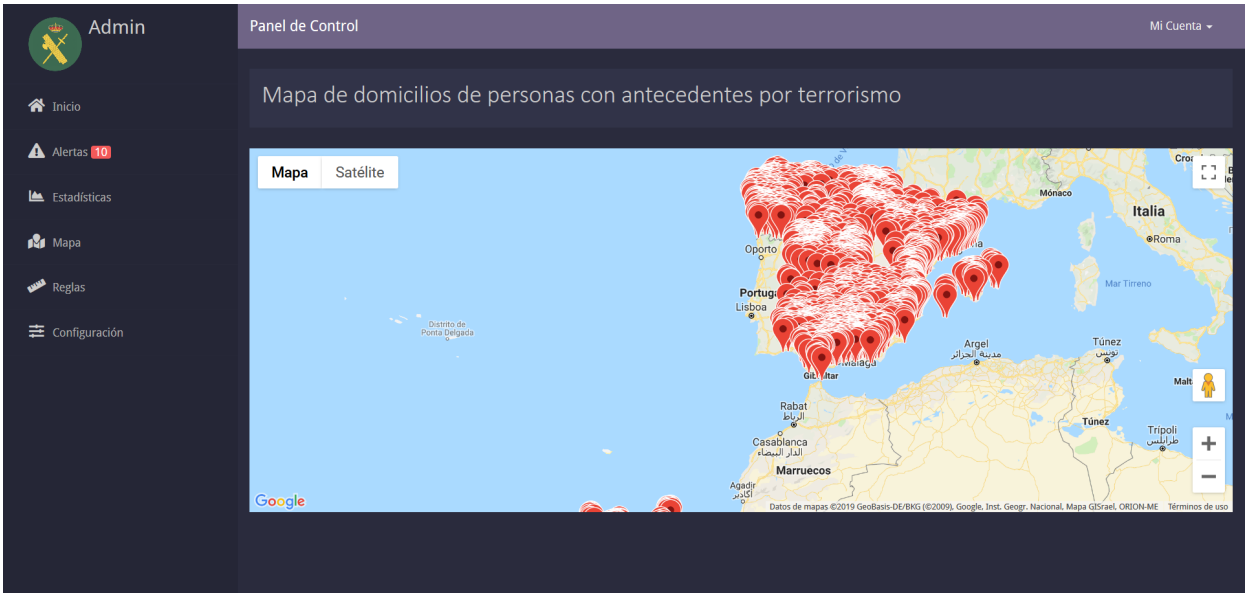


Figura 4.28: Vista mapa

Cada marca en el mapa se corresponde a un domicilio de una persona con antecedentes, y pulsando en las marcas se puede ver información sobre esa persona.

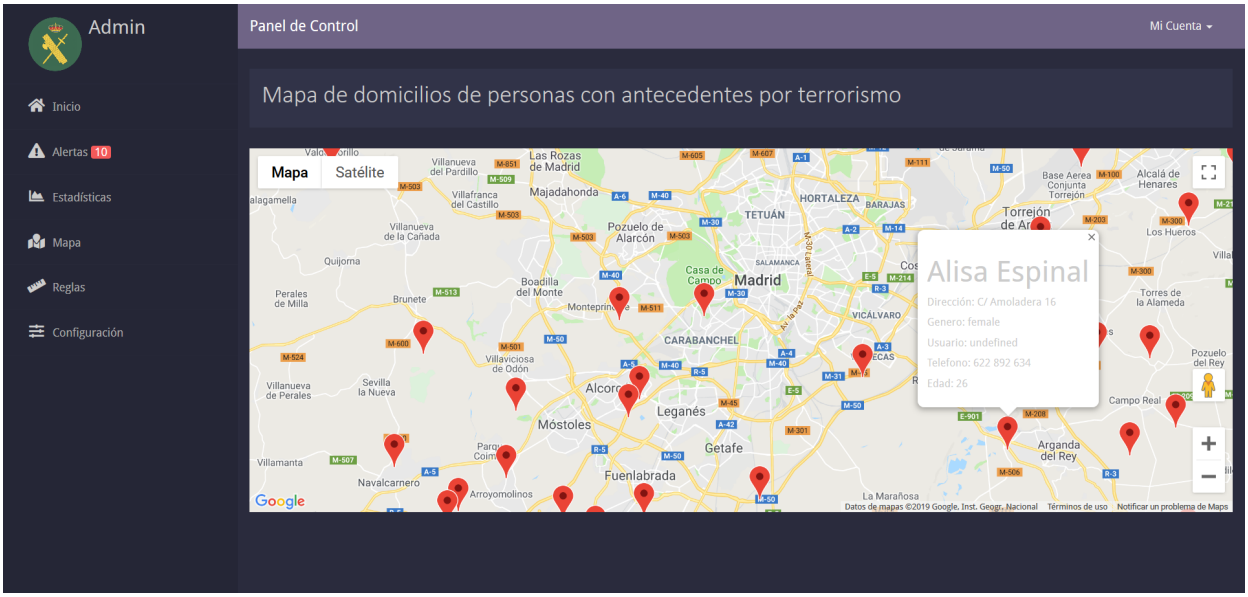


Figura 4.29: Vista mapa: información sobre personas

- `Alertas`: Sección donde se listan las alertas generadas por el Motor de Reglas (sección

4.5). Se diferencian entre alertas marcadas como resueltas y alertas pendientes. Y se ordenan por fecha situándose primero las sin resolver.

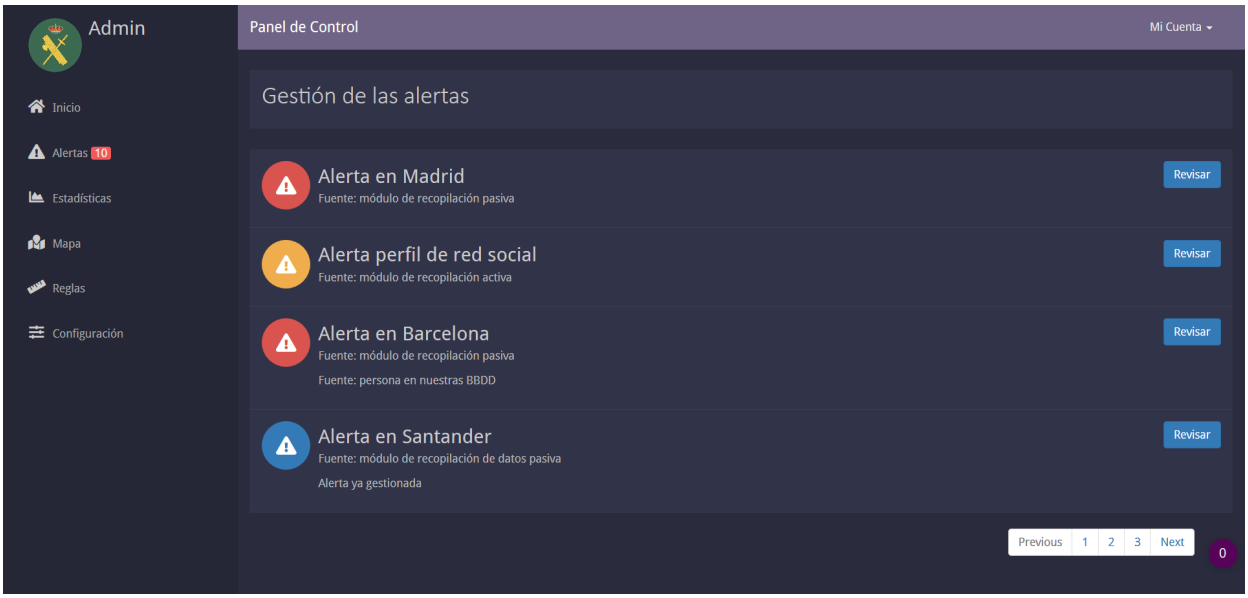


Figura 4.30: Vista alertas

- `Reglas`: Sección desde dónde se podrá acceder a la gestión y creación de las reglas.

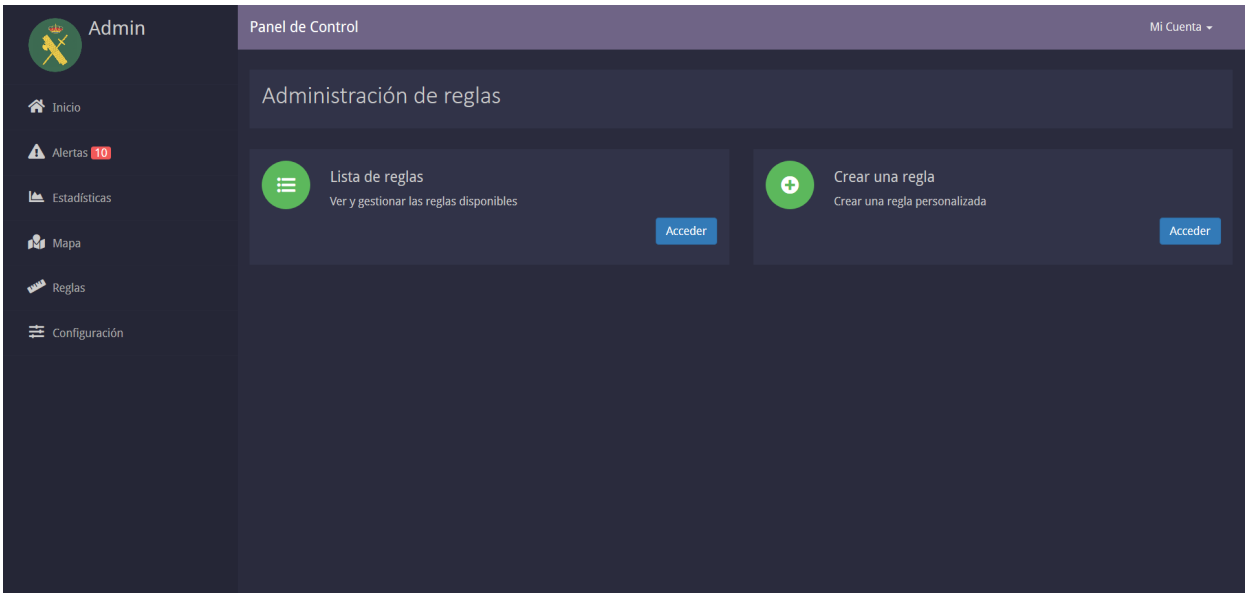


Figura 4.31: Vista reglas

- `Crear una regla`: Sección dónde se podrá crear reglas personalizadas, tanto globales, como para un usuario/persona específico.

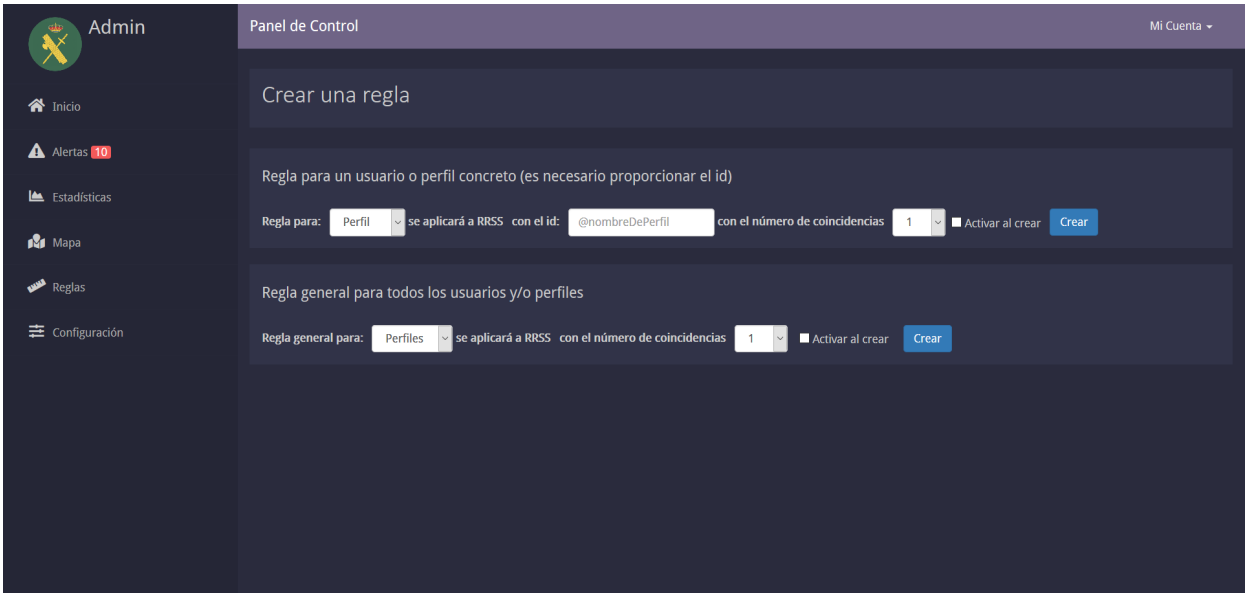


Figura 4.32: Vista creación de reglas

- `Lista de reglas`: En esta vista, se encuentran todas las reglas almacenadas en la BD. Se podrá activar/desactivar y borrar cada una de las reglas.

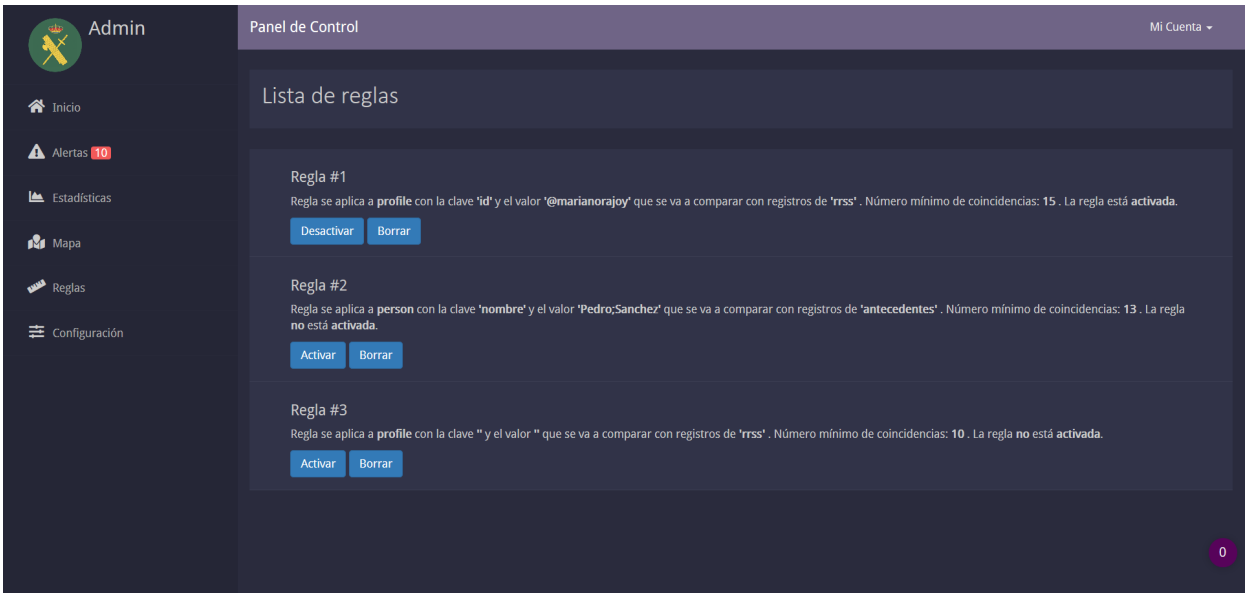


Figura 4.33: Vista listado de reglas

- `Configuración`: Sección a la cual sólo tienen acceso los usuarios con el rol de administrador. Aquí, se pueden realizar ciertas gestiones como la visualización, creación y eliminación de *hashtags* de la lista de búsqueda, o la visualización, creación, y eliminación de cuentas de usuarios.

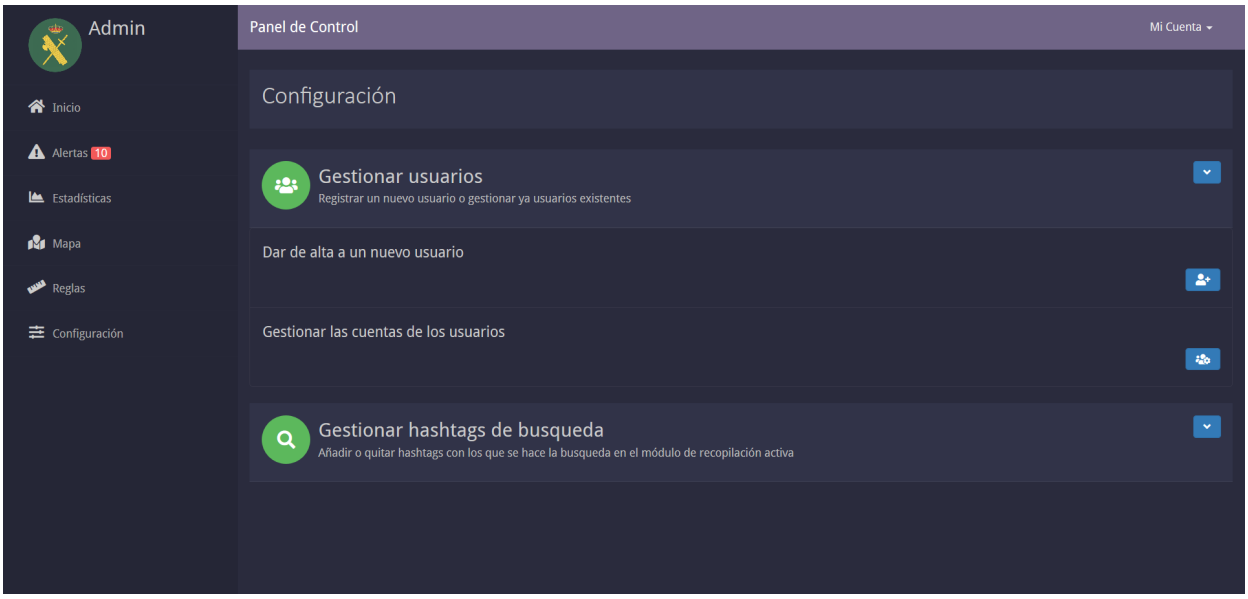


Figura 4.34: Vista configuración

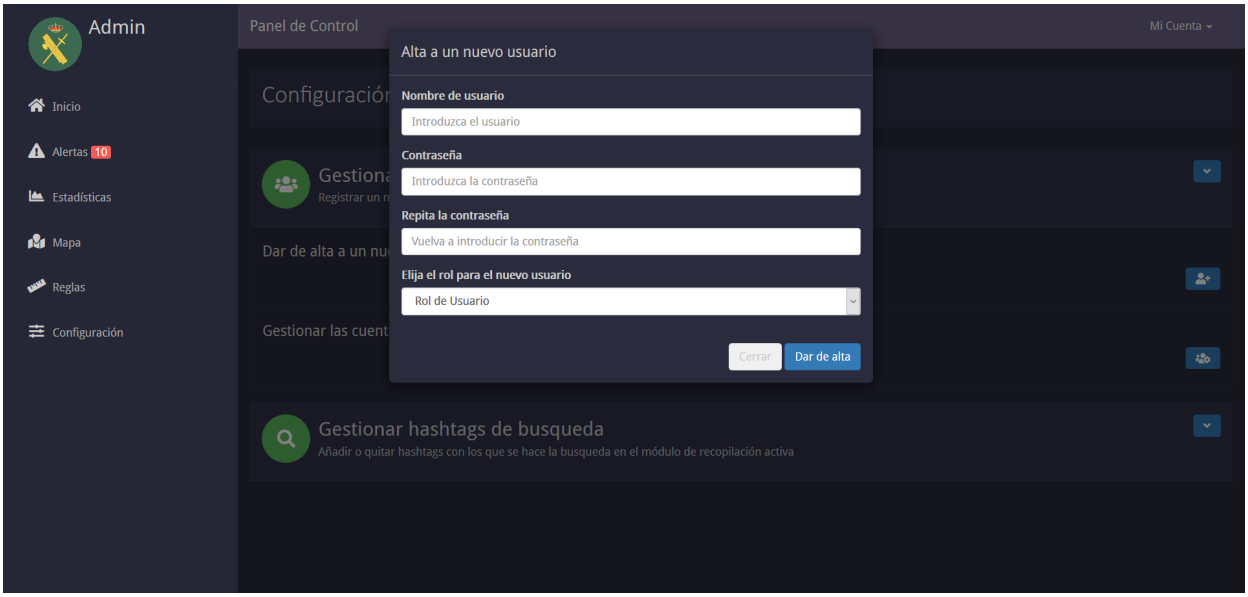


Figura 4.35: Vista alta de usuario

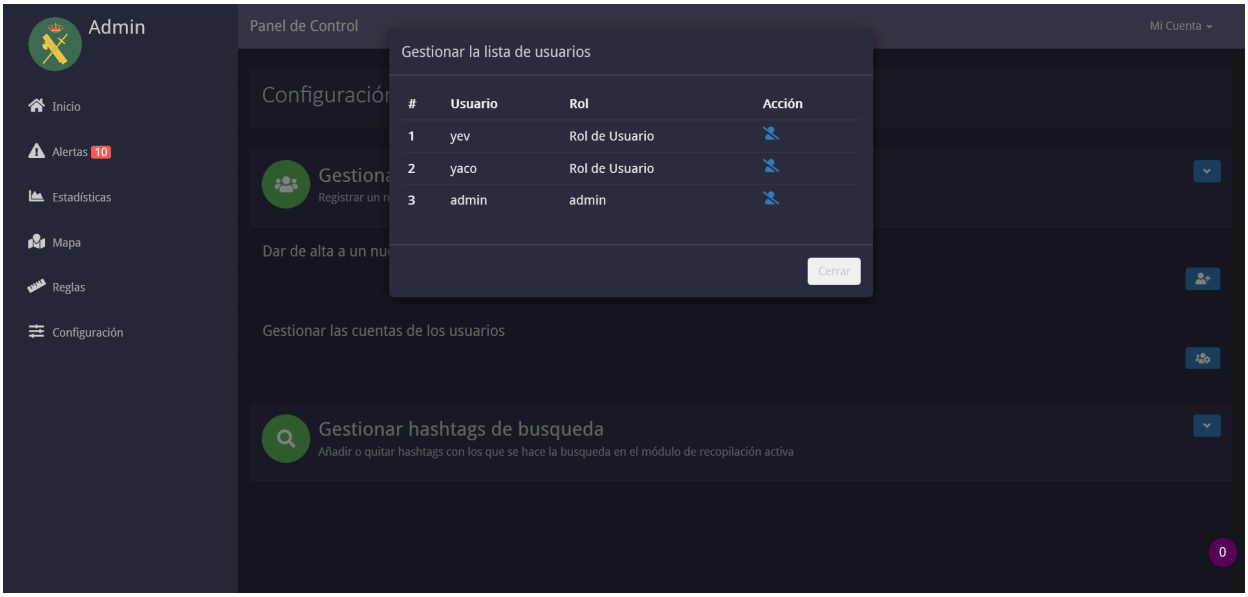


Figura 4.36: Vista gestión de usuarios

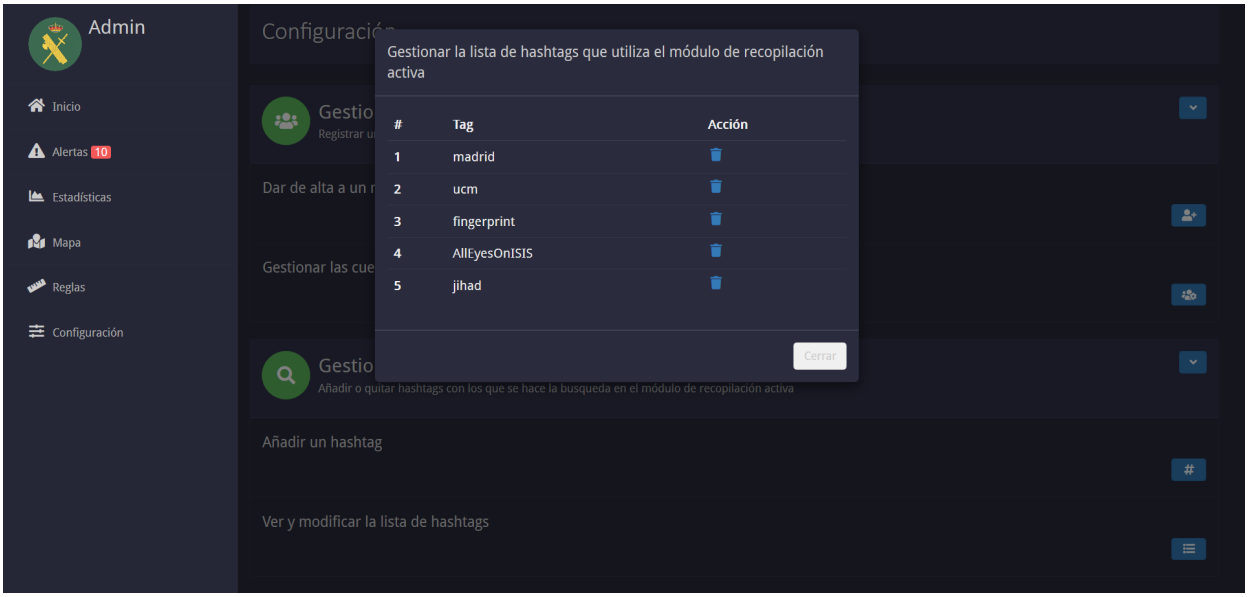


Figura 4.37: Vista gestión de hashtags

Para cada una de estas partes visuales existe un archivo JavaScript, que hace la tarea de controlador interno y da dinámica y funcionalidad a la vista.

4.6.3. Controlador

Las operaciones de esta aplicación se encuentran distribuidas, para que el código se presente de una manera más clara, reutilizable, mantenible y, a su vez, sea más modular. Hay un controlador por cada modelo de dato que se utiliza. Todas las funciones son llamadas por una petición HTTP (GET o POST). A continuación se indican sus funciones:

- `log`: Controlador que se ocupa de las operaciones referentes al modelo *log*.
  - `getStats`: Función que es llamada por una petición HTTP *GET*, que recupera de la base de datos, los elementos de la colección *logs*, para posteriormente darle formato y almacenarlo en un JSON, que se devolverá como respuesta a la petición. El objetivo es proporcionar los datos que se representarán en las vistas y en ciertos gráficos.
- `mr_Regla`: Controlador que hace las consultas referentes a la colección de reglas *mR_reglas*, que utiliza el Motor de Reglas. Cuenta con las siguientes funciones:
  - `getAll`: Función que devuelve la lista de reglas, con todos los datos de cada una de ellas.
  - `get`:Devuelve todos los datos de la regla consultada por `_id`.
  - `activate`: Función que, pasándole el `_id` de la regla en cuestión, la marca a activa, para que el Motor de Reglas la tenga en cuenta en sus ejecuciones.
  - `deactivate`: Función, que pasándole el `_id` de la regla en cuestión, la marca como desactivada para que el Motor de Reglas no la tenga en cuenta en sus ejecuciones.
  - `delete`: Función que elimina la regla seleccionada mediante el `_id`.
  - `createGlobal`: Función que crea una regla de tipo Global.
  - `createUniq`: Función que crea una regla de tipo Única.

- `person`: Controlador que hace las consultas referentes a la colección de personas *rPasiva_persons*, para la obtención de datos y estadísticas que mostrar en el *Dashboard*.

  - `getTopCities`: Función que devuelve la lista de las 5 ciudades con más personas registradas por medio de denuncias ciudadanas.

- `rActiva`: Controlador que se ocupa de las operaciones referentes a los modelos *rActivaData* y *rActivaTag*.

  - `rActiva5days`: Función que es llamada por una peticion HTTP GET, que consulta, en la base de datos, el número de publicaciones de cada red social de la colección *rActiva_Data*, para posteriormente darle formato y almacenarlo en un JSON, que se devolverá como respuesta a la petición. El objetivo es proporcionar los datos de los últimos 5 días, para que sean representados en una gráfica de barras.

  - `getAllTag`: Función que recupera de la colección *rActiva_tags* los datos referentes a todos los registros de *hashtag* de la base de datos. Esta función se utiliza para listar los *hashtags* en el panel de Configuración del *Dashboard*.

  - `delete`: Función que, pasándole el identificador, elimina de la base de datos la entrada correspondiente.

  - `addTag`: Función a la que se le pasa el *hashtag* para añadirlo a la colección *rActiva_tags*, que será utilizado por el módulo de Recopilación Pasiva para la búsqueda, de publicaciones etiquetadas, en sus próximas ejecuciones.

    Se comprueba que el *hashtag* no exista en la base de datos, y se registra.

- `record`: Controlador que se encarga de obtener la información referente a los datos de Antecedentes de la colección *records*.

  - `getAll`: Función que recupera todos los antecedentes de personas para presentarlos en un mapa geográfico.

- `getCities`: Función que recupera, de la base de datos, la lista de ciudades con el respectivo número de personas con antecedentes que residen allí, para, posteriormente, presentarlo en un mapa de actividad de puntos que indican la concentración de personas.

- `user`: Controlador que se ocupa de las operaciones referentes al modelo *user*, que gestiona junto con el *middleware* del inicio de sesión.

  - `getAll`: Función que recupera, de la colección *users*, el id, el nombre de usuario y su rol, para devolver la información como respuesta a la petición. Esta función se utiliza para listar las cuentas de usuario en el panel de Configuración.
  - `get`: Función, que pasándole el identificador del usuario, devuelve toda la información sobre él, incluida su contraseña *hasheada*.
  - `delete`: Función, que pasándole el identificador de usuario, elimina de base de datos la entrada correspondiente.
  - `register`: Función a la que se le pasan los datos (nombre, contraseña y rol) para dar de alta a un nuevo usuario. Se comprueba que no exista dicho usuario, y se registra. También, se encarga de *hashear* la contraseña mediante la biblioteca *BcryptJs* pasándola de texto plano a *hash*, para almacenarla en la base de datos con ciertas garantías de seguridad.
  - `login`: Esta función se llama cuando se pretende iniciar sesión. Comprueba si el usuario existe, y si existe, comprueba si la contraseña introducida es la correcta. Para ello, *BcryptJs* hashea la posible contraseña y la compara con el *hash* de la contraseña correspondiente al usuario. De esta manera la contraseña nunca se trata en texto plano.

4.7. Módulo extra: Importación de Datos

La clave de nuestro proyecto es el tratamiento de la información, esa información, como ya hemos explicado anteriormente, puede provenir de distintas fuentes y en distintos formatos. Por ello, hemos desarrollado este módulo de Importación de datos, que nos permitiría alimentar nuestra base de datos, no sólo con los ficheros JSON, pero también con los ficheros en formato Excel, o desde las bases de datos SQLite. Gracias a él, podemos importar de manera automatizada una gran cantidad de datos.

```
├── config/
│   └── struct.json     # Json que define la estructura de los datos
├── inputs/
│   └── BDExcel1.xls    # Excel desde el que se importan los datos
│   └── ********.xls
├── outputs/
│   └── BDExcel1.json   # Salida de los datos en formato json
│   └── ********.json
│
├── excelAjson.py       # Script de importación desde Excel
│
├── sqlite2json.py      # Script de importación desde SQLite
│
└── App.py         # Script principal
```

Figura 4.38: Estructura del módulo de importación

El módulo de Importación consiste en un programa principal, que dependiendo de los argumentos con los que se ejecuta, llama a una clase para importar los datos desde un archivo Excel, o llama a otra para importar los datos desde una base de datos SQL.

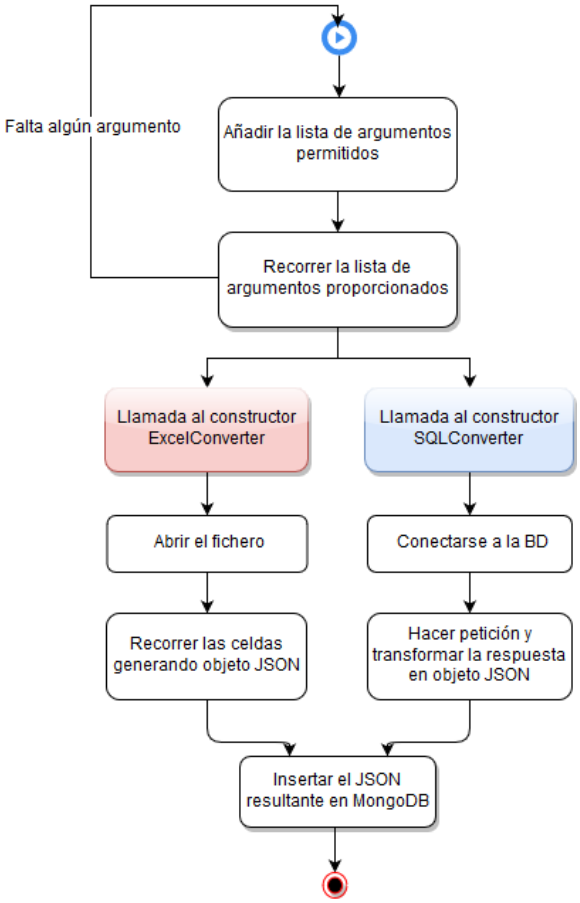


Figura 4.39: Diagrama de flujo del módulo de importación

4.7.1. Desde Excel

Su utilización solo necesita la interacción del usuario para determinar la correlación que tendrán cada fila del Excel con nuestro modelo de almacenamiento de datos.

4.7.2. Desde SQLite

Para importar los datos desde SQLite a MongoDB, hemos creado un script que hace una consulta a la base de datos SQL, y el resultado obtenido se convierte en un objeto JSON, que será almacenado en la MongoDB.

Capítulo 5. Obtención de Datos y Despliegue

El valor más importante de este proyecto es la gestión y el cruzado de la información que se realiza, pero para ello es necesario que la base de datos se nutra previamente. En este capitulo se explicarán los pasos seguidos para rellenar la base de datos ficticia de Antecedentes, donde figuran personas registradas penalmente por delitos de terrorismo.

5.1. Población de Base de Datos de Antecedentes

El sistema NDTAT necesita cotejar información con una base de datos de delitos de terrorismo, evidentemente, no disponemos de esos datos. Es por esta razón, que la hemos poblado con información ficticia generada aleatoriamente, importándola de otros medios tal y como se haría en un caso de implementación real de la solución.

5.1.1. Generación de la información

Hemos generado de manera totalmente aleatoria perfiles de personas con la ayuda de una herramienta online FakeNameGenerator[1]. Dichos datos pedimos que se exporten en formato Excel, compatible con los formatos de entrada de importación a NDTAT.

[1] https://es.fakenamegenerator.com/order.php

5.1.2. Importación de la información

En este punto entra en juego el módulo de Importación (sección 4.7). Para la importación se ha iniciado dicho módulo con el siguiente comando:

```
>python importer.py --input ./input/1.xlsx -t excel -n input1
```

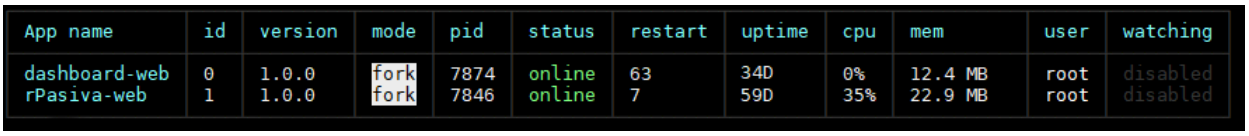
Con este comando, se ha realizado la inserción en la base de datos de todas las personas del Excel, además de obtener un archivo en formato JSON en local como respaldo de la información. Una vez hecho esto, la información ya es accesible y cotejable por el sistema mediante el Motor de Reglas.

5.2. Lanzamiento de módulos

5.2.1. Perpetuos

Los módulos perpetuos son el módulo de Recopilación Pasiva (sección 4.3) y el módulo de Representación de Datos (*Dashboard*) (sección 4.6).

Estos módulos están desarrollados en NodeJs, en local. A lo largo del desarrollo, se lanza el servicio mediante el comando `> node server.js`, pero esto tiene una serie de carencias para la puesta en producción, tales como que si se reinicia no arranca automáticamente, o que si se produce algún error, se detiene la ejecución y no se vuelve a reanudar. Por estas razones es por lo que hemos recurrido a PM2 (sección 3.10). Así podemos gestionar los lanzamientos de nuestra aplicaciones de manera unificada, disponer de *logs* y tener la seguridad de que el proceso nunca dejará de estar en ejecución.



App name	id	version	mode	pid	status	restart	uptime	cpu	mem	user	watching
dashboard-web	0	1.0.0	fork	7874	online	63	34D	0%	12.4 MB	root	disabled
rPasiva-web	1	1.0.0	fork	7846	online	7	59D	35%	22.9 MB	root	disabled

Figura 5.1: Lista de los servicios lanzados con PM2

5.2.2. Periódicos

En este caso se trata del lanzamiento de los módulos, que se van a ejecutar de manera periódica y automatizada. Para ello, hemos recurrido al comando CronTab (sección 3.8.1) de manera que definimos los siguientes periodos de tiempo:

Recopilación Activa	24h
Motor de Reglas	6h

Cuadro 5.1: Periodicidad de las ejecuciones por módulo.

Se han configurado con los siguientes comandos:

```
@midnight /bin/NDTAT/rActiva.sh
* 6 * * 1 /bin/NDTAT/motorReglas.sh
```

Dichos scripts ejecutan `python el-Modulo-En-Cuestion.py`

5.3. Seguridad en el despliegue y los accesos

5.3.1. A nivel de sistema

Todos los módulos del proyecto se encuentran en una misma máquina, esto tiene sus puntos positivos, ya que la interacción con la base de datos se realiza en local, sin pasar por servicios de terceros.

Pero sabemos que existe un peligro en desplegarlo todo en el mismo *cloud*, por ello

decidimos fortificar la máquina. Para este fin, hemos seguido la estrategia de sólo dejar acceso a servicios que son estrictamente necesarios para el funcionamiento correcto del proyecto.

Empezamos con lo básico, y lo principal, actualizar el sistema y sus servicios, ya que de poco sirven las medidas de seguridad si, por ejemplo, se sabe que la versión utilizada del sistema lleva una puerta trasera. A continuación, bloqueamos la ejecución de servicios que no son necesarios para el despliegue de los módulos, dejando el mínimo número de posibles vectores de ataque para los intrusos.

El siguiente paso fue configurar un cortafuegos, para filtrar el tráfico, tanto entrante, como saliente, a los servicios que son requeridos.

A continuación, pasamos a limitar los accesos, para ello hemos permitido el acceso únicamente por el SSH[2]. Para restringir el acceso aún más, hemos deshabilitado el login al usuario *root*, hemos prohibido el acceso con contraseña, imponiendo el acceso mediante certificados, y hemos puesto el máximo número de intentos de conexión por SSH a tres intentos. También, hemos implementado *fail2ban*[3] como protección contra ataques de fuerza bruta. Por último, hemos creado una lista blanca de direcciones IP desde las que se podrá acceder a la nube. Actualmente, en esa lista, hay solo una IP que pertenece a un servidor de OpenVPN propio y externo (Figura 5.2). Filtrar las direcciones IP durante el acceso nos permite, en todo momento, estar seguros de quién está utilizando el servicio.

[2] `https://www.ssh.com/`
[3] `https://www.fail2ban.org/wiki/index.php/Main_Page`

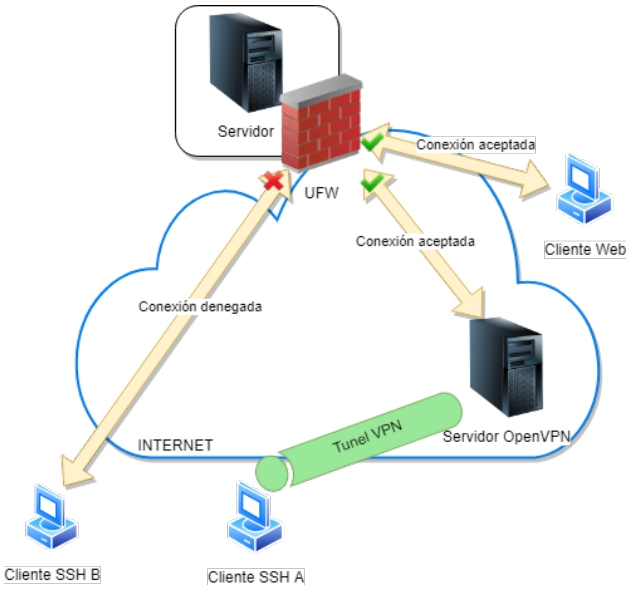


Figura 5.2: Esquema de Conexión

También lo hemos tenido en cuenta, de cara a la base de datos, securizando las conexiones y creando un usuario con privilegios únicamente de consulta (selección, inserción, actualización y borrado).

5.3.2. A nivel de aplicación

El módulo de Visualización de los datos, o el *Dashboard*, filtra las direcciones IP entrantes, permitiendo solo aquellas que se encuentran en la lista blanca. Aparte, para acceder al *Dashboard* los usuarios tienen que iniciar sesión obligatoriamente y, en función del rol que tengan, tendrán acceso a unas u otras secciones.

Capítulo 6. Casos de uso

En este capítulo vamos a presentar distintos casos de uso de nuestro proyecto. Para este fin, hemos inventado casos ficticios, cualquier parecido con alguno de los hechos ocurridos en la realidad es mera coincidencia.

6.1. Caso de uso: módulo de colaboración ciudadana

Hace mucho tiempo, en una galaxia muy, muy lejana, las fuerzas de la Alianza Rebelde consiguieron derrotar al Imperio e instaurar la República. Los imperialistas, los pocos que quedan, no aceptaron la derrota y se organizaron en pequeños grupos, para seguir luchando como guerrillas, haciendo atentados en barrios céntricos de las ciudades republicanas. Para intentar prevenir esos atentados, la Alianza implantó el sistema NDTAT como una herramienta de ayuda a sus fuerzas de seguridad.

Así, un día normal, los habitantes de un pueblo cercano a una de las metrópolis más grandes de la República, vieron un movimiento extraño en una casa que llevaba años abandonada. Unos vecinos fueron testigos de la aparición de vehículos de carga, nunca vistos en el pueblo hasta ese día, otros vieron como unos individuos desconocidos descargaban lo que parecían ser bombonas de gas. Estos hechos alarmaron a algunos de los habitantes, por lo cual decidieron usar la aplicación de Colaboración Ciudadana informando sobre lo ocurrido.

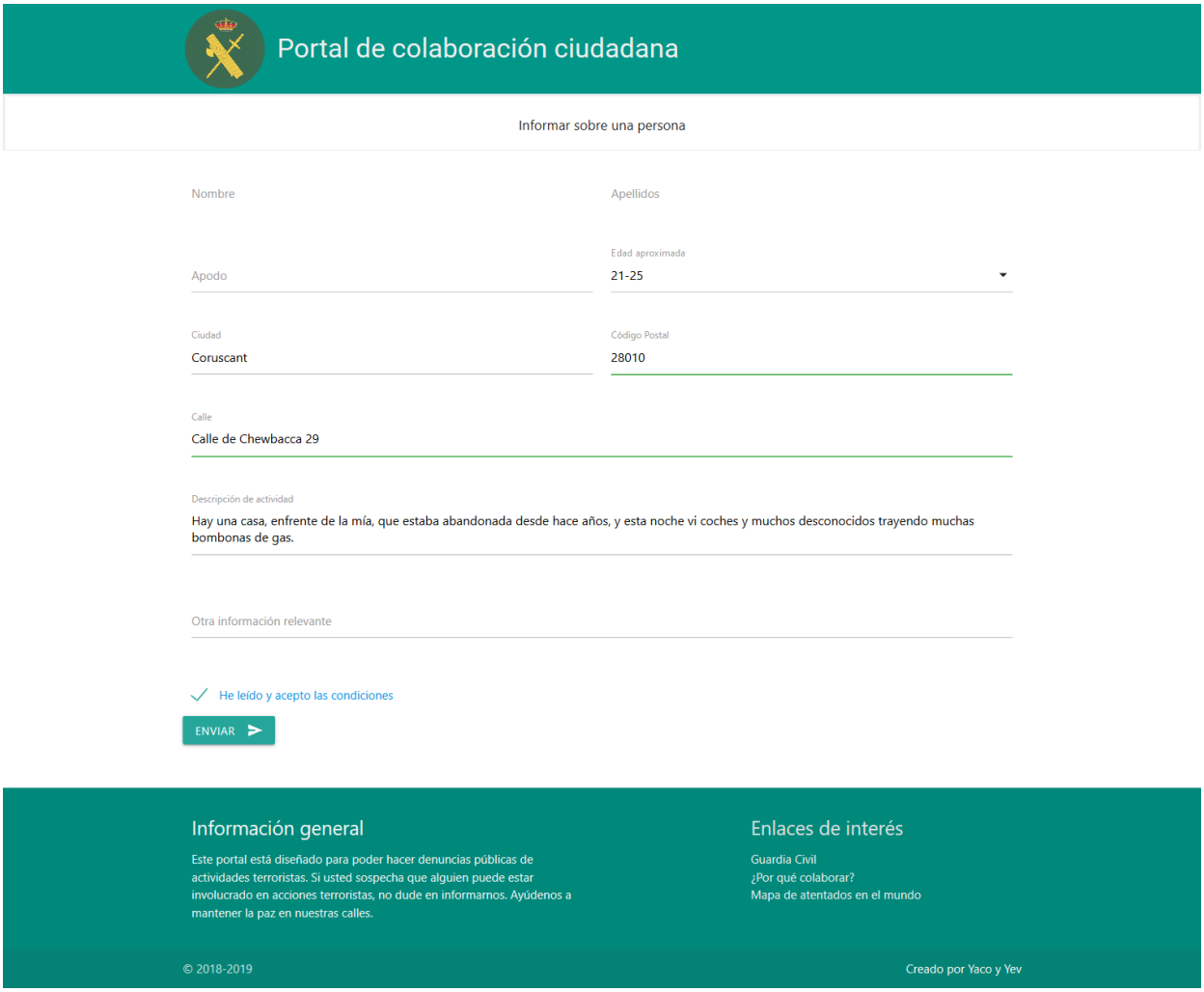


Figura 6.1: Caso de Uso: Reporte de uno de los vecino

En ese momento, un agente de la Guardia Jedi, estuvo revisando las alertas generadas por el sistema de reglas y se encontró con una nueva alerta proveniente de ese pueblo. Sin pensarlo dos veces, el agente solicitó a un escuadrón que vaya a revisar esa casa.

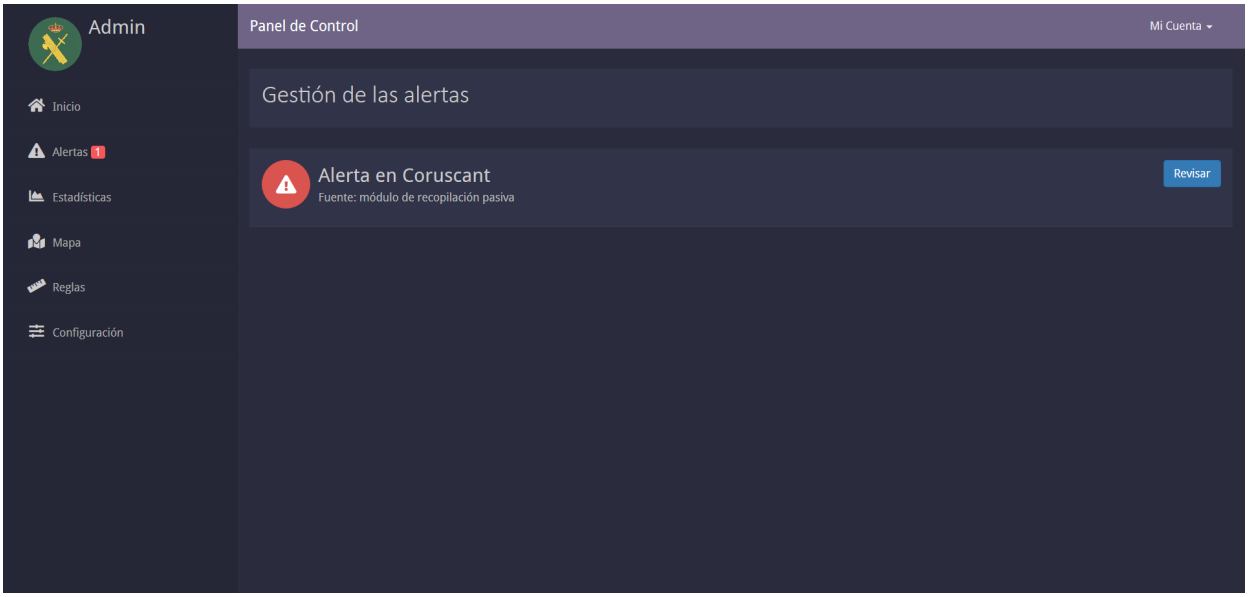


Figura 6.2: Caso de Uso: Nueva alerta en Dashboard

Cuando llegaron los agentes, se encontraron con un laboratorio improvisado de fabricación de explosivos y cinco personas dentro, que resultaron ser miembros de una de las organizaciones de los imperialistas, todos han sido detenidos. Esta intervención ha salvado muchas vidas previniendo el futuro atentado.

6.2. Caso de uso: módulo de recopilación activa

Durante los últimos años, internet se ha convertido en una parte indispensable de la vida de los ciudadanos de la República. Incluso los más jóvenes, a día de hoy, pasan más tiempo *twitteando* sobre las últimas carreras de pods, que interactuando con otros jóvenes en la vida real.

Las organizaciones de los imperialistas también se han dado cuenta de la importancia de internet y de su poder a la hora de divulgar sus ideologías. Cada poco tiempo crean campañas para conseguir más adeptos para su causa, y lo hacen en las redes sociales como

Instagram y Twitter, donde los usuarios son más jóvenes y más propensos a ser reclutados. Una de las campañas más famosas y virales fue #AllEyesOnEmpire, cuando los imperialistas empezaron a subir multitud de imágenes desde diferentes puntos de la Galaxia. Aparte de las campañas, los imperialistas crean perfiles para mostrar a los jóvenes ómo es la vida de un combatiente de su ideología y lo hacen de una manera muy tentadora.

La Guardia Jedi, teniendo la herramienta NDTAT, puede hacer seguimiento automatizado de las publicaciones, y en caso de que una misma cuenta haga un número de publicaciones mayor al definido por el motor de reglas, se genera una alerta. Y así pasó, una nueva campaña de reclutamiento organizada por los imperialistas se ha podido frenar a tiempo, informando a los equipos de las redes sociales sobre las actividades delictivas.

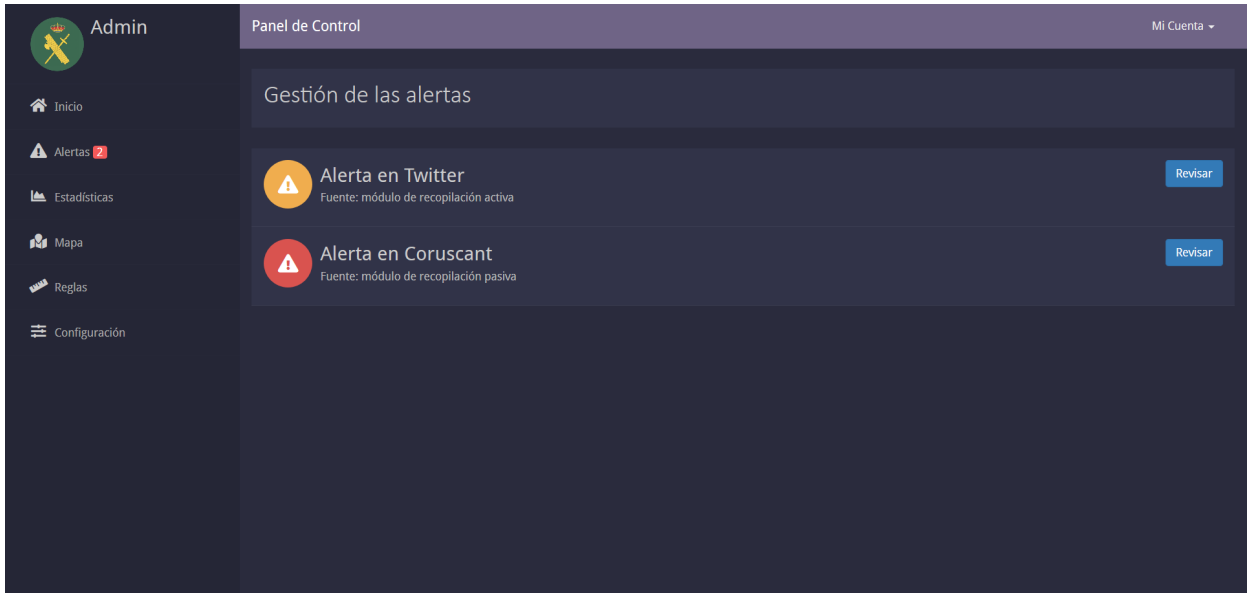


Figura 6.3: Caso de Uso: Alerta RRSS

Capítulo 7. Conclusiones

Desde que empezamos este Trabajo Fin de Grado, se han perpetrado más de mil setecientos atentados por todo el mundo[1], con más de siete mil víctimas en un solo año. El objetivo principal de nuestro proyecto, desde el principio, ha sido aportar un nuevo enfoque a la lucha contra estos sucesos, creando una herramienta automatizada que pueda servir de ayuda para las Fuerzas y Cuerpos de Seguridad del Estado. Hasta el momento, no se conoce, públicamente, ninguna herramienta parecida a la Nube de Detección Temprana de Actividades Terroristas, lo que convierte nuestro proyecto en algo único.

NDTAT está creado como un sistema completamente modular, pensando en los futuros desarrollos y haciendo que implementar y conectar nuevos módulos sean tareas rápidas y sencillas. Además, una vez desplegada la nube y configurados todos sus componentes, apenas es necesaria la interacción de una persona.

En vista a este proyecto, hemos desarrollado cinco módulos. El módulo de Recopilación Pasiva de Datos consiste en un portal de colaboración ciudadana, donde los ciudadanos pueden aportar información sobre cualquier suceso relacionado con el terrorismo. El módulo de Recopilación Activa de Datos, busca y almacena las publicaciones que tengan que ver con el terrorismo en las redes sociales. Un módulo de Importación de Datos, que facilita la alimentación de la Base de Datos permitiendo utilizar ficheros de distintos tipos. El Motor

[1] `https://storymaps.esri.com/stories/terrorist-attacks/`

de Reglas analiza los datos según las reglas aplicadas. Y por último, el módulo de Visualización de Datos, o el *Dashboard*, donde se podrá encontrar las estadísticas de todos los datos, visualizar un mapa con las concentraciones de la gente con antecedentes por terrorismo, y crear y gestionar las reglas, los *hashtags* a analizar, los usuarios del sistema y mucho más.

Personalmente, somos conscientes de que el trabajo desarrollado está bien, teniendo en cuenta el límite de tiempo para realizarlo. También, sabemos que quedarían bastantes cosas por hacer, para que el proyecto pueda pasar de ser una prueba de concepto a un producto final. Creemos que, invirtiendo más tiempo y más medios, podría tener mucho potencial para una aplicación real en los sistemas de las FFCCSE. Aún así, estamos muy contentos con lo conseguido.

Chapter 7. Conclusions

Since we started this Final Degree Project, there have been perpetrated more than one thousand and seven hundred terrorist attacks, with more than seven thousand of victims in the last year. From the beginnings, the goal of our project was to find a new approach to fighting against terrorism, creating an automatized tool that could be of help to the State Security Forces. There is not publicly known alternatives to NDTAT, so it makes this project something unique.

NDTAT is made as a modular system, with the idea of being quickly deployed and easily complemented by future developments and modules. Also, once deployed, NDTAT barely needs the interaction of a human, because mostly all of the processes are automatized.

For this project, we developed five modules. The passive data gatherer, that consists in a web application for citizen collaboration, where the citizens can give any information about any terrorism related events. The active data gatherer, searches for tweets and Instagram publications with connection to terrorism. The importation module, is meant to ease the importation process, allowing the data input from different file types. The rules engine analyzes all information according to the provided rules. And finally, the dashboard module, where you can find the data statistics, display the people with terrorist background concentration map, create and manage the rules, hashtags, users and more.

Sincerely, we realize that we have done a good work, considering the time limit.

However, we also know, that there are a lot of things left to do, in order to make this project evolve from beeing a proof of concept to a final product. We think that with more time and resource dedication, this project could have a big potential for real application in the State Security Forces systems. Even so, we are happy with our achievements.

Capítulo 8. Reparto del Trabajo

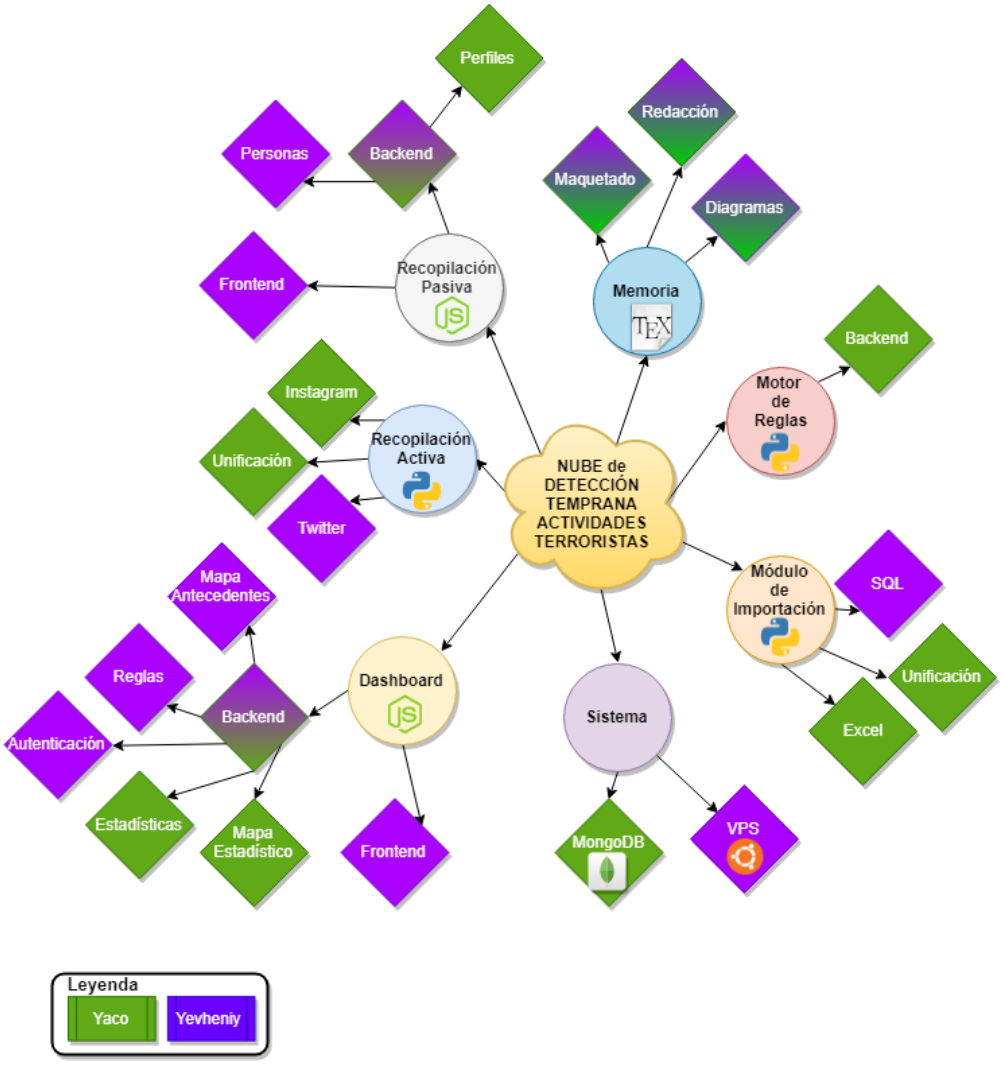


Figura 8.1: Esquema general de Reparto del Trabajo.

8.1. Aportación de Yaco

Mis aportaciones a este trabajo han consistido, principalmente, en la parte del *backend* de los distintos módulos. A continuación procederé a listar y explicar brevemente las tareas que he llevado a cabo.

- **Memoria**: En mi caso he realizado la redacción de apartados como la sección de arquitectura de la base de datos, la composición de los diagramas de las colecciones y modelos JSON, la sección del Motor de Reglas, parte de la sección del *backend* del *Dashboard*, la sección de importación de datos, el manual de usuario y el proceso de importación de datos.
- **Sistema**: A nivel de sistema me he encargado de la base de datos. Llevando a cabo tareas como:
  - Inicialmente, la gestión de la base de datos en el *cloud* de Mongodb y, posteriormente instalación, despliegue y configuración de Mongodb en el *VPS* realizando la configuración y la securización de la misma.
  - La de definición de los modelos de datos de la base de datos y de que manera están distribuidos en las distintas colecciones
  - La integración inicial de los módulos para que empezasen a recuperar la información desde la Base de Datos
  - La generación y posterior importación de los antecedentes a la Base de Datos
  - La exportación de las colecciones para su entrega junto al código de cara a la entrega.
- **Recopilación Pasiva**: Este fue el primer módulo que desarrollamos, y aporté mis conocimientos previos de NodeJs haciendo la estructura del *backend* en cuanto al

servidor y al enrutado de las direcciones *REST*, y realicé lo referente al modelo de dato, el controlador de Mongoose de perfiles y parte de lo referente a reportes.

- **Recopilación Activa**: En este módulo nos repartimos el trabajo por cada una de las redes sociales, en mi caso me ocupé del desarrollo de la recopilación de publicaciones desde Instagram. Mi tarea consistió en obtener mediante *Web Scrapping* las publicaciones referentes a un *hashtag* para posteriormente recorrerlas y almacenar los datos relevantes en la base de datos.

  Una vez realizado los desarrollos de recopilación para Twitter e Instagram, disponíamos de dos scripts con ejecuciones independientes, por lo que había que realizar una unificación y pequeñas modificaciones para combinarlos en un único script que realizase ambas tareas, y una estandarización de los datos obtenidos y objetos generados para su inserción en la colección *rActiva_data* con un modelo común.

- **Modulo de importación**: El desarrollo de este módulo tuvo una planificación similar a la del inmediatamente anterior. Nos repartimos la importación de formato SQL y formato Excel, en mi caso consistió en el desarrollo de la importación desde el formato de *Microsoft*, y la posterior unificación en un mismo script unificando el código con el script de importación SQL.

- **Dashboard**: En este caso, mi aportación consistió en todo el desarrollo referente a las estadísticas. Tanto en su registro en los diferentes módulos como en su presentación en la pantalla principal y en los gráficos generados con *Google Charts*. La representación de densidad de personas con antecedentes sobre el mapa estadístico mediante consultas concreta y las agregaciones para obtener la información necesaria de la bd. También me encargué de la parte *backend* del inicio de sesión y registro de usuarios integrando el *hasheo* de contraseñas.

- **Motor de Reglas**: La idea y especificaciones del funcionamiento del Motor de Reglas fue un común entre los dos, y posteriormente realicé el desarrollo del Motor de Reglas

en su totalidad. Este desarrollo ha consistido en la lógica necesaria para que sea posible aplicar todas las reglas definidas en el sistema para cada uno de los nuevos perfiles y personas que se hayan registrado en las ultimas horas y la generación y registro de alertas.

8.2. Aportación de Yevheniy

- **Memoria**: En cuanto a memoria, hice la redacción de los siguientes apartados: introducción, con su versión en inglés, resumen, con su versión en inglés, la sección de objetivos, la sección de plan de trabajo, la sección de estado del arte, parte de tecnologías, la sección de arquitectura global del proyecto, la parte correspondiente a *frontend* del módulo de Recopilación Pasiva, la parte de Twitter del módulo Recopilación Pasiva, parte correspondiente a la vista del módulo de Recopilación Activa, parte del módulo de Importación, la sección de Seguridad en el despliegue y los accesos, el capítulo de Casos de uso, el capítulo de las Conclusiones, con su versión en inglés y el capítulo de Trabajo futuro.

- **Sistema**: En cuanto a sistema, me he encargado del despliegue de los módulos en el *VPS* y de la securización del *cloud*, donde pude aportar mis conocimientos de ciberseguridad.

- **Recopilación Pasiva**: Para el módulo de Recopilación Pasiva, me ocupe del desarrollo de *frontend*, y de una parte de *backend* creando el controlador de Mongoose de personas, y una pequeña parte de lo referente a reportes.

- **Recopilación Activa**: En cuanto al módulo de Recopilación Activa, me ocupé de la parte de Twitter. Mi tarea consistió en crear un programa que vaya recopilando las publicaciones de esa red social, haciendo una búsqueda por *hashtags*, para almacenarlas en la base de datos

- **Modulo de Importación**: En cuanto a este módulo, mi tarea fue desarrollar un script para importar los datos, desde una base de datos de tipo SQLite, para su posterior uso en el Motor de Reglas.

- **Dashboard**: En cuanto a *Dashboard*, me ocupe de la creación de las siguientes tareas:

  - Creación de vista Login
  - Creación de la vista Inicio
  - Creación de la vista Estadísticas
  - Creación de la vista Mapa
  - Creación de la vista Reglas
  - Creación de la vista Configuración
  - La lógica de gestión de usuarios
  - La lógica de gestión de *hashtags*
  - La lógica de gestión de reglas
  - La lógica de la vista del Mapa
  - Control de acceso y sesiones

  Cabe destacar, que la parte de representación de estadísticas, tanto gráficos en la vista de Estadísticas, como la información mostrada en Inicio, es desarrollada por Yaco.

Capítulo 9. Trabajo futuro

En esta sección presentaremos todas las ideas que creemos que podrían seguir enriqueciendo el proyecto y que nos hemos visto obligados a dejar en el tintero, debido a que supondrían un aumento del alcance, o del tiempo de desarrollo que supera las expectativas de un TFG de un año.

- Let's Encrypt[1]: la idea de nuestro proyecto es crear un sistema completo, tanto a nivel de funcionalidades, como a nivel de seguridad. En la sección 5.3, ya explicamos las medidas de seguridad aplicadas, pero nos faltaría una, no menos importante que las citadas: segurizar las comunicaciones *HTTP* mediante los certificados *SSL/TLS*. Para ello, hemos pensado en utilizar la autoridad de certificación *Let's Encrypt*, respaldada por multitud de empresas y organizaciones, que permite crear dichos certificados de manera gratuita y automatizada. El motivo, por el cual no hemos podido utilizar esta tecnología, es que los certificados sólo pueden ser generados para los dominios, y no para las direcciones IP.

- Crear un algoritmo de análisis de sentimientos para el módulo de recopilación activa. Ahora mismo guardamos todas las publicaciones encontradas por los *hashtags*, sin previo procesamiento. Sin embargo, implementar un algoritmo de análisis sentimental nos ayudaría a quitar muchos de los falsos positivos.

[1] `https://letsencrypt.org/`

Este tipo de análisis es una de las técnicas de procesamiento de lenguaje natural, por ello hemos pensado utilizar la librería NLTK[2] de Python, considerada un referente en el mundo de minería de datos. Esta técnica consiste en separar el texto de la publicación en palabras, analizar la connotación positiva o negativa de cada palabra, y así determinar la polaridad contextual de un usuario respecto al tema en cuestión.

- Crear una red neuronal para que nos ayude a detectar los perfiles en las redes sociales, relacionados con el terrorismo. Para ello, sería necesario crear un *dataset* de publicaciones, de un tamaño considerable, de diferentes cuentas terroristas. Para ello, escogeríamos cuentas de terroristas ya condenados, o perfiles encontrados por los analistas.

- Utilizar la API de Telegram[3], para crear un bot que funcione como un *crawler* de grupos y usuarios. Telegram es una aplicación de mensajería multiplataforma, que cada vez está ganando más popularidad, sobre todo por la seguridad y privacidad que proporciona al usuario. Esto tiene también tiene un lado negativo, que la aplicación es muy utilizada por organizaciones terroristas, y por ese motivo algunos países quieren bloquear Telegram[4]. Por todo ello, pensamos que sería interesante crear un bot, que vaya recopilando grupos y usuarios, siguiendo los enlaces que se publican en los canales de temática terrorista.

- Creación de un *Shell Script* que realice de manera automática los pasos necesarios para el despliegue por primera vez de la aplicación, configurando las tareas cron y las instancias de PM2.

[2] `https://www.nltk.org/`
[3] https://core.telegram.org/
[4] `https://www.nytimes.com/2018/05/02/world/europe/telegram-iran-russia.html`

Capítulo 10. Manual

Hemos realizado unos breve manual para la utilización y despliegue de la aplicación. En este manual, se enumerarán los prerrequisitos del sistema para el lanzamiento de NDTAT, y se darán las indicaciones necesarias para ello.

10.1. Prerrequisitos del sistema

Será necesario instalar o tener instalado los siguientes binarios:

- Python 3 (sección 3.1) y su gestor de paquetes *pip*.
- NodeJs (sección 3.4) y su gestor de paquetes *npm*.
- MongoDB (sección 3.5), y en caso de no querer interactuar con la base de datos mediante la linea de comando, un gestor de bases de datos MongoDB.

Una vez se tiene instalados, se procederá a la instalación de las librerías y paquetes necesarios para el funcionamiento de los módulos.

10.2. Instalaciones de paquetes

- En el caso de los módulos desarrollados en NodeJs, será más sencillo, simplemente hay que ejecutar el comando `>npm install` en las carpetas de los módulos de Recopilación Pasiva y *Dashboard*.
- En el caso de los módulos desarrollados en Python, hay que realizar la instalación uno a uno.

10.2.1. Base de datos

Se deberán importar las colecciones de la BD, y crear un usuario con permisos *CRUD* con el nombre `app`.

10.3. Lanzamiento de los módulos

- Si se va a lanzar en un entorno de desarrollo o pruebas, se puede lanzar con el comando `> node server.js` en cada proyecto NodeJs, de no ser así y querer lanzarlo en un entorno de producción se recomienda instalar PM2(3.10) y lanzar las ejecuciones mediante:

```
>pm2 start server.js --name "dasboard-web"
>pm2 start server.js --name "rPasiva-web"
```

- En el caso de los módulos desarrollados en Python, como se ha explicado en la sección 5.2, recomendamos definir un cron (sección 3.8.1) para su ejecución periódica. Si se tratase de un entorno de desarrollo o pruebas se podría lanzar mediante:

```
>python aplicacion.py
```

Apéndice A. Carta de Recomendación

MINISTERIO
DEL INTERIOR

GUARDIA CIVIL
DIRECCIÓN GENERAL

Secretaría de Cooperación Internacional

A quien pueda interesar,

El Oficial que suscribe, que participó como enlace entre el equipo investigador y la Guardia Civil, en el proyecto denominado "Nuevos algoritmos algebraicos y estocásticos para los servicios de información en la lucha contra el terrorismo: Desarrollo de un software de *apoyo a la toma de decisiones"*, desarrollado durante los años 2014 y 2015 en el Centro Universitario de la Defensa de Zaragoza,

CERTIFICA QUE:

El trabajo fin de grado realizado por los estudiantes de la Facultad de Informática de la Universidad Complutense de Madrid Yevheniy Kushch y Yaco Alejandro Santiago Pérez, titulado "Nube de Detección Temprana de Actividades Terroristas", está alineado con las actividades desarrolladas por la Guardia Civil y responde a unas necesidades reales de la misma.

En Madrid, a 29 de abril de 2019

DIRECCIÓN GENERAL DE LA GUARDIA CIVIL - SECI -

Pedro Antonio López Castelló
Comandante de la Guardia Civil
Secretaria de Cooperación Internacional

Apéndice B. Glosario

Por orden de aparición, vamos a listar palabras y siglas que puedan necesitar explicación:

- NDTAT: Nube de Detección Temprana de Actividades Terroristas
- FFCCSE: Fuerzas y Cuerpos de Seguridad del Estado
- OSINT: Open Source INTelligence
- POO: Programación orientada a objetos
- Web Scraping: Extracción de información de sitios web simulando ser un navegador web
- BD: Base de datos
- RRSS: Redes Sociales
- Dashboard: Hace referencia al portal web donde se representan los datos y las estadísticas generadas
- MVC: Patrón de diseño Modelo Vista Controlador
- Framework: Entorno que tiene funciones predefinidas con el fin de facilitar tareas facilitar el desarrollo

- Frontend: Parte visual del sitio web con la que interactúan los clientes
- Backend: Parte de la algoritmia y funciones del código en el lado del servidor
- CSV: Formato de archivo de almacenamiento de información separado por comas
- Hasheada: Información que ha se ha obtenido tras calcular el hash de la información inicial mediante formulas matemáticas
- Middleware: Clase que interviene en el intercambio de información
- HTTP: Protocolo de transmisión web. HTTPS su versión segura
- SSL: Capa de seguridad en la transmisión de información entre cliente y servidor web
- Dataset: Conjunto de datos recopilados
- Cron: Demonio de linux que se encarga de ejecutar procesos con cierta regularidad definida previamente
- Crawler: Rastreador web para recopilar información
- Shell Script: Script de comandos e instrucciones para linux
- CRUD: Operaciones en base de datos de inserción, selección, actualización y borrado

Bibliografía

[1] R. L. Corral, "El uso de las nuevas tecnologıas por el terrorismo yihadista", *Cuadernos de la Guardia Civil*, 2017, https://intranet.bibliotecasgc.bage.es/intranet-tmpl/prog/local_repository/documents/19075.pdf.

[2] I. R. Alonso, "Terrorismo yihadista y nuevas tecnologías", *Crimipedia*, 2016, http://crimina.es/crimipedia/wp-content/uploads/2016/07/CRIMIPEDIA_TERRORISMO-Y-NUEVAS-TECNOLOG%C3%8DAS_ISABEL-RAMOS.pdf.

[3] S. Vanden Broucke y B. Baesens, *Practical web scraping for data science: best practices and examples with Python*. Apress, 2018.

[4] E. Martin, *Blog con abundante información sobre web scraping y casos concretos de Instagram*, https://edmundmartin.com/.

[5] M. Omar, *Ejemplo de Dashboard*, https://codepen.io/MustafaOmarIbrahim/full/jLMPKm.

[6] M. Pilgrim y S. Willison, *Dive Into Python 3*. Springer, 2009, vol. 2.

[7] D. Herron, *Node Web Development*. Packt Publishing Ltd, 2013.

[8] S. Tilkov y S. Vinoski, "Node. js: Using JavaScript to build high-performance network programs", *IEEE Internet Computing*, vol. 14, n.º 6, págs. 80-83, 2010.

[9] K. Chodorow, *MongoDB: the definitive guide: powerful and scalable data storage*. .º'Reilly Media, Inc.", 2013.

Yevheniy Kushch - Yaco Alejandro Santiago Pérez - 2019